O'REILLY®




图灵程序设计丛书

第2版

# 统计思维

## 程序员数学之概率统计

Think Stats

[美] Allen B. Downey 著

金迎 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

# 数字版权声明

# 译者介绍

## 金迎

1997年毕业于北京大学计算机系。从事软件开发工作数年。2004年毕业于中科院计算所计算机应用技术专业，之后进入软件测试行业。具有丰富的手工和自动化测试的项目经验。


图灵程序设计丛书

# 统计思维

## 程序员数学之概率统计（第2版）

# Think Stats

## Second Edition

[美] Allen B. Downey 著
金迎 译

O'REILLY®

*Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo*

O'Reilly Media, Inc.授权人民邮电出版社出版

人 民 邮 电 出 版 社
北 京

图书在版编目（ＣＩＰ）数据

统计思维 ： 程序员数学之概率统计 ： 第2版 / (美)
唐尼 (Downey,A.B.) 著 ; 金迎译. -- 2版. -- 北京 :
人民邮电出版社, 2015.9
（图灵程序设计丛书）
ISBN 978-7-115-40108-3

Ⅰ. ①统… Ⅱ. ①唐… ②金… Ⅲ. ①概率统计
Ⅳ. ①O211

中国版本图书馆CIP数据核字(2015)第176735号

## 内 容 提 要

这是一本以全新视角讲解概率统计的入门书。抛开经典的数学分析，Downey 手把手教你用编程理解统计学。具体说来，本书通过一个案例研究，介绍探索性数据分析的全过程：从收集数据、生成统计信息，到发现模式、验证假设。同时研究分布、概率规则、可视化和其他多种工具及概念。此外，第 2 版新增了回归、时间序列分析、生存分析和分析方法等章节。

本书既适合作为教材，又适合作为程序员学习概率统计的参考书，也适合作为非程序员了解概率统计与编程的工具书。

◆ 著　　　[美] Allen B. Downey
译　　　金　迎
责任编辑　岳新欣
执行编辑　张　庆
责任印制　杨林杰

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　印刷

◆ 开本：800×1000　1/16
印张：12.75
字数：302千字　　　　2015年 9 月第 2 版
印数：9001－13 000册　　　　2015年 9 月北京第 1次印刷
著作权合同登记号　图字：01-2015-2497号

定价：49.00元
**读者服务热线：(010)51095186转600　印装质量热线：(010)81055316**
**反盗版热线：(010)81055315**
**广告经营许可证：京崇工商广字第 0021 号**

# 版权声明

# O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

## 业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

# 目录

前言 ······ xi

第 1 章　探索性数据分析 ······ 1

1.1　统计学方法 ······ 2
1.2　全国家庭增长调查 ······ 2
1.3　数据导入 ······ 3
1.4　DataFrame ······ 4
1.5　变量 ······ 6
1.6　数据变换 ······ 6
1.7　数据验证 ······ 8
1.8　解释数据 ······ 9
1.9　练习 ······ 10
1.10　术语 ······ 11

第 2 章　分布 ······ 13

2.1　表示直方图 ······ 14
2.2　绘制直方图 ······ 14
2.3　全国家庭增长调查中的变量 ······ 15
2.4　离群值 ······ 18
2.5　第一胎 ······ 18
2.6　分布概述 ······ 20
2.7　方差 ······ 21
2.8　效应量 ······ 21
2.9　报告结果 ······ 22
2.10　练习 ······ 23
2.11　术语 ······ 23

第 3 章　概率质量函数……25

3.1　概率质量函数……25
3.2　绘制 PMF……26
3.3　绘制 PMF 的其他方法……28
3.4　课堂规模悖论……29
3.5　使用 DataFrame 进行索引……31
3.6　练习……33
3.7　术语……34

第 4 章　累积分布函数……35

4.1　PMF 的局限……35
4.2　百分位数……36
4.3　CDF……37
4.4　表示 CDF……38
4.5　比较 CDF……39
4.6　基于百分位数的统计量……40
4.7　随机数……41
4.8　比较百分位秩……42
4.9　练习……43
4.10　术语……44

第 5 章　分布建模……45

5.1　指数分布……45
5.2　正态分布……48
5.3　正态概率图……49
5.4　对数正态分布……51
5.5　Pareto 分布……53
5.6　随机数生成……56
5.7　为什么使用模型……56
5.8　练习……57
5.9　术语……59

第 6 章　概率密度函数……61

6.1　PDF……61
6.2　核密度估计……63
6.3　分布框架……65
6.4　Hist 实现……65
6.5　Pmf 实现……66
6.6　Cdf 实现……67
6.7　矩……68

6.8 偏度 69
6.9 练习 72
6.10 术语 73

第 7 章 变量之间的关系 75

7.1 散点图 75
7.2 描述关系特征 78
7.3 相关性 79
7.4 协方差 80
7.5 Pearson 相关性 81
7.6 非线性关系 82
7.7 Spearman 秩相关 82
7.8 相关性和因果关系 83
7.9 练习 84
7.10 术语 85

第 8 章 估计 87

8.1 估计游戏 87
8.2 猜测方差 89
8.3 抽样分布 90
8.4 抽样偏倚 93
8.5 指数分布 93
8.6 练习 95
8.7 术语 95

第 9 章 假设检验 97

9.1 经典假设检验 97
9.2 假设检验 98
9.3 检验均值差 100
9.4 其他检验统计量 101
9.5 检验相关性 102
9.6 检验比例 103
9.7 卡方检验 104
9.8 再谈第一胎 105
9.9 误差 106
9.10 功效 107
9.11 复现 108
9.12 练习 109
9.13 术语 109

第 10 章 线性最小二乘法 111
10.1 最小二乘法拟合 111
10.2 实现 112
10.3 残差 113
10.4 估计 114
10.5 拟合优度 116
10.6 检验线性模型 118
10.7 加权重抽样 119
10.8 练习 121
10.9 术语 121
第 11 章 回归 123
11.1 StatsModels 124
11.2 多重回归 125
11.3 非线性关系 127
11.4 数据挖掘 128
11.5 预测 129
11.6 Logistic 回归 131
11.7 估计参数 132
11.8 实现 133
11.9 准确度 134
11.10 练习 135
11.11 术语 136
第 12 章 时间序列分析 139
12.1 导入和清洗数据 139
12.2 绘制图形 141
12.3 线性回归 143
12.4 移动平均值 144
12.5 缺失值 146
12.6 序列相关 148
12.7 自相关 149
12.8 预测 150
12.9 参考书目 154
12.10 练习 154
12.11 术语 155
第 13 章 生存分析 157
13.1 生存曲线 157
13.2 危险函数 159

13.3 估计生存曲线 160
13.4 Kaplan-Meier 估计 161
13.5 婚姻曲线 162
13.6 估计生存函数 163
13.7 置信区间 164
13.8 群组效应 166
13.9 外推 168
13.10 预期剩余生存期 169
13.11 练习 171
13.12 术语 172

第 14 章 分析方法 173

14.1 正态分布 173
14.2 抽样分布 174
14.3 表示正态分布 175
14.4 中心极限定理 176
14.5 检验 CLT 177
14.6 应用 CLT 180
14.7 相关检验 181
14.8 卡方检验 183
14.9 讨论 184
14.10 练习 184

作者介绍 186

封面介绍 186

# 前言

本书介绍探索性数据分析的实用工具，书中章节按照我自己处理数据集时遵循的步骤进行组织。

- 导入和清洗：无论数据格式如何，我们通常都需要花费一些时间和精力进行数据的读取、清洗和变换，并进行检查，以确保在此过程中信息完好无损。
- 单变量探索：通常情况下，我会首先逐个检查变量，弄清变量的意义，分析变量值的分布，选择合适的汇总统计量。
- 成对探索：为了发现变量之间的关系，我会分析表格和散点图，计算相关性并进行线性拟合。
- 多变量分析：如果变量之间存在明显关系，我就要使用多元回归以增加控制变量，从而研究更复杂的关联关系。
- 估计和假设检验：在汇报统计结果时，有 3 个重要问题需要回答。效应规模如何？再次运行同一测量时，预期的变化性有多大？这个明显的效应是否可能是偶然产生的？
- 可视化：在数据探索中，可视化是寻找可能关系和效应的一个重要工具。如果一个明显的效应是统计显著的，那么可视化可以帮助我们有效地展示结果。

本书采用的是计算方法。相比数学方法，计算方法具有如下优点。

- 大多数概念用 Python 代码进行展示，而非数学符号。总体而言，Python 代码的可读性更好，而且这些代码是可执行的，读者可以下载、运行并进行修改。
- 每一章都附有练习，可以帮助读者扩展并巩固知识。编写程序时，你把自己对知识的理解表达为代码；调试代码时，这些理解也可以得到修正。
- 一些练习使用了实验检验统计行为。例如，你可以通过生成随机样本并计算它们的总和来探索中心极限定理（Central Limit Theorem，CLT）。练习得到的可视化结果展示了 CLT 的工作原理及适用条件。

- 一些概念很难从数学角度进行理解，却很容易通过模拟掌握。例如，通过运行随机模拟对 $p$ 值进行近似，可以增强我们对 $p$ 值含义的理解。
- 由于本书使用通用编程语言（Python），因此读者几乎可以从任何数据源导入数据，而不必受限于使用特定统计工具进行了清洗和格式化的数据集。

本书使用基于项目的方法。在我的课堂上，学生需要完成一个为期一个学期的项目。在项目中，学生要提出一个统计问题，寻找可以解决这个问题的数据集，并将学到的各种技术应用于这个数据集。

为了展示我采用的统计分析方法，本书将介绍一个贯穿各章的案例。这个案例使用的数据来自以下两方面资源。

- 全国家庭增长调查（National Survey of Family Growth，NSFG），这一调查由美国疾病控制和预防中心（Center for Disease Control and Prevention，CDC）开展，以收集“与家庭生活、婚姻状况、妊娠情况、生育情况、避孕情况，以及两性健康相关的信息”。参见 http://cdc.gov/nchs/nsfg.htm。
- 行为危险因素监测系统（Behavioral Risk Factor Surveillance System，BRFSS），由国家慢性病预防和健康促进中心（National Center for Chronic Disease Prevention and Health Promotion）主持，以“跟踪美国的健康状况及风险行为”。参见 http://cdc.gov/BRFSS/。

其他示例使用的数据来自美国国税局（IRS）、美国人口普查（U.S. Census）及波士顿马拉松赛（Boston Marathon）。

《统计思维》的第 2 版包含了第 1 版的各章，但对其中很多内容进行了大幅修改，并新增了关于回归、时间序列分析、生存分析和分析方法的章节。本书第 1 版没有使用 pandas、SciPy 和 StatsModels，所以这些内容也都是新增的。

# 写作思路

人们在编写新教材时，通常会参考已有教材。这样做的结果就是，大部分图书都采用相似的结构顺序叙述相似的内容。

我没有这样做。实际上，在撰写本书时出于以下几点考虑，我几乎没有使用任何纸质资料。

- 我的目的是探索新方法，因此不想过多介绍已有方法。
- 既然本书的授权是免费的，那么我希望书中所有的内容都不受版权限制。
- 我的很多读者都无法到提供纸质资料的图书馆去，因此我尽量引用互联网上的免费资源。

- 一些传统媒体的支持者认为，只使用电子资料是偷懒且不可靠的做法。关于偷懒，他们可能说对了，但是我不认为电子资料不可靠，因此希望对自己的理论进行验证。

我使用最多的资源是维基百科（Wikipedia）。总的来说，我在维基百科上读到的统计资料都很不错（但是我在后续也做了一些小的改动）。本书多处引用了维基百科页面，希望你能通过提供的链接阅读这些资料。很多时候，维基百科页面是本书内容的补充。除去我认为必要的修改，书中使用的术语和符号与维基百科基本一致。另外两个我觉得有用的资源是 Wolfram Mathworld 和 Reddit 统计论坛（http://www.reddit.com/r/statistics）。

# 本书代码

本书使用的代码和数据都可从 GitHub（htts://github.com/AllenDoweny/ThinkStats2）下载。Git 是一个版本管理系统，可以对项目文件进行跟踪。受 Git 管理的文件集称为*代码库*（repository）。GitHub 是一项托管服务，可以存储 Git 代码库，并提供一个便于使用的 Web 接口。

我的 GitHub 主页提供以下几种使用代码的方法。

- 你可以点击 Fork 按钮，在 GitHub 上创建该代码库的副本。如果你还没有 GitHub 账号，就需要创建一个。创建副本之后，你就在 GitHub 上拥有了自己的代码库，可以跟踪学习本书时编写的代码。之后你可以复制这个代码库，即将文件复制到自己的计算机上。
- 或者，你也可以复制我的代码库。这一操作不需要 GitHub 账号，但是你对代码所做的修改无法写回 GitHub。
- 如果你完全不想使用 Git，那么可以点击 GitHub 页面右下角的按钮，下载文件的 Zip 包。

本书所有代码都无需翻译即可在 Python 2 和 Python 3 中直接运行。

编写本书代码时，我使用的是 Continuum Analytics 的 Anaconda，这是一个免费的 Python 版本，其中带有运行本书代码所需的所有软件包（还有很多其他包）。Anaconda 很容易安装。默认情况下，Anaconda 进行用户级而非系统级安装，因此不需要管理员权限。Anaconda 同时支持 Python 2 和 Python 3，你可以从 Continuum（http://continuum.io/downloads）进行下载。

如果你不想使用 Anaconda，那么需要安装以下软件包。

- pandas，进行数据的表示和分析。下载地址为：http://pandas.pydata.org/。
- NumPy，支持基本的数字运算。下载地址为：http://www.numpy.org/。

- SciPy，进行科学计算，包括统计运算。下载地址为：http://www.scipy.org/。
- StatsModels，进行回归分析和其他统计分析。下载地址为：http://statsmodels.sourceforge.net/。
- matplotlib，支持可视化。下载地址为：http://matplotlib.org/。

虽然这些都是常用软件包，但并不是所有的 Python 安装都包含这些包，而且在有些环境下很难进行安装。如果你无法安装这些包，我强烈建议你使用 Anaconda，或者包含这些包的其他 Python 版本。

当你复制完代码库或者将 Zip 包解压后，会得到一个名为 ThinkStats2/code 的文件夹，其中有一个 nsfg.py 文件。运行 nsfg.py 会读取一个数据文件，运行一些测试，并输出一条消息，如“All tests passed”。如果你得到的是 import error，可能是因为缺少某些必要的软件包。

本书的大部分练习都使用 Python 脚本，但也有一些使用 IPython 记事本。如果你之前没有用过 IPython 记事本，可以访问文档 http://ipython.org/ipython-doc/stable/notebook/notebook.html 得到帮助。

本书读者应该熟悉 Python 的核心功能，包括面向对象的特征，但无需具备 pandas、NumPy 和 SciPy 知识。如果你已经熟知这些模块，可以跳过一些相关小节。

本书读者应该了解基本的数学知识，例如对数和求和。本书中有几处会涉及微积分概念，但你无需进行微积分运算。

如果你从未学习过统计学，本书会是一本很好的入门教材。如果你学习过传统的统计学课程，那么我希望本书能够修正你过去接受的一些错误观点。

—

Allen B. Downey 是一位计算机科学教授，执教于美国马萨诸塞州尼德姆的富兰克林欧林工程学院。

# 致谢

如果你有任何建议或者更正，请发送电子邮件至 downey@allendowney.com。如果我采纳了你的意见并对书中内容进行了修改，会将你的名字会加入致谢列表（除非你拒绝这样做）。

请在邮件中给出存在错误的句子，或句子的一部分，以便我进行搜索。当然，提供页码和章节号也可以，但还是以句子内容为佳。谢谢！

- Lisa Downey 和 June Downey 阅读了本书初稿，作出了很多更正，也提出了很多建议。
- Steven Zhang 发现了数处错误。

- Andy Pethan 和 Molly Farison 帮助调试了一些解决方案，Molly 还发现了数处拼写错误。
- Andrew Heine 发现了 error 函数中的一个错误。
- Dr. Nikolas Akerblom 知道一只始祖马有多大。
- Alex Morrow 对一个代码示例进行了说明。
- Jonathan Street 在关键时刻发现了一个错误。
- Gábor Lipták 发现了本书的一处拼写错误及接力赛问题的解决方案。
- 非常感谢 Kevin Smith 和 Tim Arnold 设计了 plasTeX，让我可以将本书转为 DocBook。
- George Caplan 的建议使本书结构更加清晰。
- Julian Ceipek 发现了一处错误和很多拼写错误。
- Stijn Debrouwere、Leo Marihart III、Jonathan Hammler 和 Kent Johnson 更正了本书第 1 印次中的错误。
- Dan Kearney 发现了一处拼写错误。
- Jeff Pickhardt 发现了一个损坏的链接和一处拼写错误。
- Jörg Beyer 发现了书中的拼写错误，并对书中代码的帮助文档进行了许多修正。
- Tommie Gannert 发送了一个补丁文件，其中包括许多更正。
- Alexander Gryzlov 对一个练习中的说明提出了建议。
- Martin Veillette 报告了一个 Pearson 相关性公式中的一处错误。
- Christoph Lendenmannt 提交了数个勘误。
- Haitao Ma 发现了一处拼写错误，并告诉了我。
- Michael Kearney 提出了很多极佳的建议。
- Alex Birch 提出了一些很有益的建议。
- Lindsey Vanderlyn、Griffin Tschurwald 和 Ben Small 阅读了本书的早期版本，并发现了很多错误。
- John Roth、Carol Willing 和 Carol Novitsky 进行了技术审阅，发现了很多错误，并提出了许多有益的建议。
- Rohit Deshpande 发现了一处排版错误。
- David Palmer 提出了很多建议，作出了不少更正。
- Erik Kulyk 发现了很多拼写错误。

## Safari® Books Online




Safari Books Online 是应需而变的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。

Safari Books Online 是技术专家、软件开发人员、Web 设计师、商务人士和创意人士开展调研、解决问题、学习和认证培训的首选资料。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。

成为 Safari Books Online 的会员，你即可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBMRedbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，请访问我们的网站（http://www.safaribooksonline.com）。

# 联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：

http://bit.ly/think_stats_2e

对于本书的评论和技术性问题，请发送电子邮件到：bookquestions@oreilly.com。

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：http://facebook.com/oreilly。

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia。

我们的 YouTube 视频地址如下：http://www.youtube.com/oreillymedia。

# 第1章 探索性数据分析

如果能将数据与实际方法相结合，就可以在存在不确定性时解答问题并指导决策，这就是本书的主题。

举个例子。我的妻子在怀第一胎时，我听到了一个问题：第一胎是不是经常晚于预产期出生？下面所给出的案例研究就是由这个问题引出的。

如果用谷歌搜索这个问题，会看到大量的讨论。有人认为第一胎的生产日期确实经常晚于预产期，有人认为这是无稽之谈，还有人认为恰恰相反，第一胎常常会早产。

在很多此类讨论中，人们会提供数据来支持自己的观点。我发现很多论据是下面这样的。

> “我有两个朋友最近都刚生了第一个孩子，她们都是超过预产期差不多两周才出现临产征兆或进行催产的。”
>
> “我的第一个孩子是过了预产期两周才出生的，我觉得第二个孩子可能会早产两周！”
>
> “我认为这种说法不对，因为我姐姐是头生子，而且是早产儿。我还有好些表兄妹也是这样。”

这些说法都是基于未公开的数据，通常来自个人经验，因此称为轶事证据（anecdotal evidence）。在闲聊时讲讲轶事当然无可厚非，所以我并不是要批评以上那几个人。

但是，我们可能需要更具说服力的证据以及更可靠的回答。如果按照这个标准进行衡量，轶事证据通常都靠不住，原因有如下几点。

- 观测值数量较小

  如果第一胎的孕期的确偏长，这个时间差与正常的偏差相比可能很小。在这种情况下，我们可能需要比对大量的孕期数据，才能确定这种时间差确实存在。

- 选择数据时存在偏倚

  人们之所以参与这个问题的讨论，有可能是因为自己的第一个孩子出生较晚。这样的话，这个选择数据的过程就会对结果产生影响。

- 确认数据时存在偏倚

  赞同这种说法的人也许更可能提供例子进行佐证。持怀疑态度的人则更可能引用反例。

- 不精确

  轶事通常都是个人经验，经常会记错、误传或者误解等。

那我们该如何更好地回答这个问题呢？

## 1.1 统计学方法

为了解决轶事证据的局限性，我们将使用以下统计学工具。

- 数据收集

  我们将使用大型的全国性调查数据，这个调查专门设计用于对美国人口进行有效的统计推断。

- 描述性统计

  得出统计量，对数据进行简要的汇总，并评估可视化数据的不同方法。

- 探索性数据分析

  寻找各种模式、差异，以及其他能够解决我们感兴趣的问题的特征，同时还将检查数据的不一致性，发现局限性。

- 估计

  使用样本数据来估计一般总体的统计特征。

- 假设检验

  如果看到明显的效应，例如两个群组之间存在差异，将衡量该效应是否是偶然产生的。

谨慎执行上面的步骤，并避免各种错误，我们就可以获得合理性和准确性更高的结论。

## 1.2 全国家庭增长调查

从 1973 年起，美国疾病控制和预防中心（CDC）就开始进行全国家庭增长调查（NSFG，

http://cdc.ogv/nchs/nsfg.htm），以收集“与家庭生活、婚姻状况、妊娠情况、生育情况、避孕情况，以及两性健康相关的信息。此项调查的结果用于……进行健康服务和健康教育项目的规划，以及对家庭、生育及健康情况进行统计研究”。

我们将使用这项调查收集到的数据研究第一胎是否出生较晚，并解答一些其他问题。为了有效地使用这些数据，我们必须理解这项研究是如何设计的。

全国家庭增长调查是一项*横截面*（cross-sectional）研究，也就是说该研究捕获的是一个群组在某一时刻的快照。在横截面研究之外，最常见的是*纵向*（longitudinal）研究，指在一个时间段内重复观察一个群组。

全国家庭增长调查进行过 7 次，每一次都称为一个*周期*（cycle）。我们将使用第 6 次的数据，其时间段为 2002 年 1 月至 2003 年 3 月。

这项调查的目的是对一个*总体*（population）得出结论。全国家庭增长调查的目标总体是居住在美国、年龄在 15~44 岁的人。理想情况下，调查要收集这个总体中每个成员的数据，但这是不可能实现的。实际上，我们收集了这个总体的一个子集的数据，这个子集称为*样本*（sample）。参与调查的人称为*调查参与者*（respondent）。

通常来说，横截面研究应该是*有代表性*（representative）的，也就是说目标总体中每个成员参与调查的机会均等。这种理想条件在实践中很难实现，但是进行调查的人员会竭尽所能满足这个条件。

全国家庭增长调查不具有代表性，而是特意进行*过度抽样*（oversample）。这项研究的设计者招募了拉美裔美国人、非洲裔美国人和青少年 3 个群组的参与者，每个群组的招募比例都超过其在美国人口中所占的比例，以确保各群组的参与者数量足够多，从而进行有效的统计推断。

当然，过度抽样也有缺点，那就是不容易从调查的统计数据中得出关于总体的结论。我们稍后会对此进行讨论。

在使用这种调查数据时，我们必须熟悉*代码本*（codebook），这一点非常重要。代码本记录了一项研究的设计、使用的调查问题，以及调查中响应变量的编码。你可以从美国疾病控制和预防中心的网站（http://www.cdc.gov/nchs/nsfg/nsfg_cycle6.htm）下载全国家庭增长调查数据的代码本和使用手册。

## 1.3　数据导入

本书所用的代码和数据都可以通过 GitHub（https://github.com/AllenDowney/ThinkStats2）获取。前言中介绍了如何下载和使用这些代码。

下载代码后，你会得到一个名为 ThinkStats2/code 的文件夹，其中包含一个名为 nsfg.py

的文件。运行 nsfg.py 会读取数据文件，执行测试，然后打印出一条消息，例如“All test passed”。

让我们看看这个文件所执行的工作。第 6 次全国家庭增长调查的妊娠数据保存在名为 2002FemPreg.dat.gz 的文件中，这是一个纯文本（ASCII 码）形式的 gzip 压缩文件，有固定宽度的列。这个文件中的每一行都是一个*记录*（record），包含一次妊娠的数据。

2002FemPreg.dct 是一个 Stata 字典文件，记录了数据文件的格式。Stata 是一个统计软件。Stata“字典”是由变量名、变量类型及标识变量位置的索引值组成的列表。

下面几行摘自 2002FemPreg.dct：

```
infile dictionary {
  _column(1)  str12  caseid    %12s  "RESPONDENT ID NUMBER"
  _column(13) byte   pregordr   %2f  "PREGNANCY ORDER (NUMBER)"
}
```

这个字典描述了两个变量：`caseid` 是一个长度为 12 的字符串，代表调查参与者的 ID；`pregorder` 是一个单字节整数，说明这条记录描述的是这位调查参与者的第几次妊娠。

下载的代码包含一个 thinkstats2.py 文件，这是一个 Python 模块，包含了本书中用到的很多类和函数，其中有读取 Stats 字典和全国家庭增长调查数据文件的函数。这两个函数在 nsfg.py 中的用法如下：

```
def ReadFemPreg(dct_file='2002FemPreg.dct',
                dat_file='2002FemPreg.dat.gz'):
    dct = thinkstats2.ReadStataDct(dct_file)
    df = dct.ReadFixedWidth(dat_file, compression='gzip')
    CleanFemPreg(df)
    return df
```

`ReadStataDct` 的参数是字典文件名，返回值 `dct` 是一个 `FixedWidthVariables` 对象，其中包含从字典文件中得到的信息。`dct` 对象提供 `ReadFixdWidth` 方法进行数据文件的读取。

## 1.4 DataFrame

`ReadFixedWidth` 方法返回一个 DataFrame 对象。DataFrame 是 pandas 提供的基础数据结构。pandas 是一个 Python 数据和统计包，它的使用会贯穿本书。在 DataFrame 中，每个记录为一行（在我们的例子中就是每个妊娠数据为一行），每个变量为一列。

除了数据，DataFrame 还包含变量名和变量类型信息，并提供访问和修改数据的方法。

如果打印 `df` 对象，你会看到其中行列的部分数据和 DataFrame 的大小：13 593 行 / 记录，244 列 / 变量。

```
>>> import nsfg
>>> df = nsfg.ReadFemPreg()
>>> df
...
[13593 rows x 244 columns]
```

`df` 的 `columns` 属性将列名返回为一列 Unicode 字符串。

```
>>> df.columns
Index([u'caseid', u'pregordr', u'howpreg_n', u'howpreg_p', ... ])
```

`df.columns` 的结果是一个 Index 对象，Index 也是一个 pandas 数据结构。我们稍后会详细介绍 Index，现在可以暂时将其视为一个列表。

```
>>> df.columns[1]
'pregordr'
```

要访问 DataFrame 中的一列，你可以将列名作为键值。

```
>>> pregordr = df['pregordr']
>>> type(pregordr)
<class 'pandas.core.series.Series'>
```

其结果是一个 Series 对象，这又是一个 pandas 数据结构。Series 与 Python 列表类似，还能提供一些附加功能。打印一个 Series 对象会得到索引和对应的数值。

```
>>> pregordr
0     1
1     2
2     1
3     2
...
13590    3
13591    4
13592    5
Name: pregordr, Length: 13593, dtype: int64
```

这个示例中的索引是从 0 到 13 592 的整数，但通常索引可以使用任何可排序的数据类型。这个示例中的元素也是整数，但元素可以是任何类型的。

示例中的最后一行列出了变量名、Series 长度和数据类型。`int64` 是 NumPy 提供的类型之一。如果在 32 位机器上运行这个示例，得到的数据类型可能是 `int32`。

你可以使用整数的 index 和 slice 值访问 Series 中的元素。

```
>>> pregordr[0]
1
>>> pregordr[2:5]
2    1
3    2
4    3
Name: pregordr, dtype: int64
```

index 操作符的结果是 `int64`，slice 的结果还是一个 Series。

你也可以使用点标记法来访问 DataFrame 中的列。

```
>>> pregordr = df.pregordr
```

只有当列名为合法的 Python 标识符时（即以字母开头，不包含空格等），才能使用这种写法。

## 1.5 变量

我们已经使用了全国家庭增长调查数据集中的两个变量——`caseid` 和 `pregordr`，还看到数据集中共有 244 个变量。本书的探索性分析用到如下变量。

- `caseid`：调查参与者的整数 ID。
- `prglength`：妊娠周数，是一个整数。
- `outcome`：怀孕结果的整数代码。1 代表成功生产。
- `pregordr`：妊娠的顺序号。例如，一位调查参与者的第一次妊娠为 1，第二次为 2，以此类推。
- `birthord`：成功生产的顺序号，一位调查参与者的第一个孩子代码为 1，以此类推。对没有成功生产的其他妊娠结果，此字段为空。
- `birthwgt_lb` 和 `birthwgt_oz`：新生儿体重的磅部分数值和盎司部分数值。
- `agepreg`：妊娠结束时母亲的年龄。
- `finalwgt`：调查参与者的统计权重。这是一个浮点数，表示这位调查参与者在全美人口中代表的人数。

如果你仔细阅读了代码本，就会发现这些变量中很多都是*重编码*（recode），也就是说这些不是调查收集的*原始数据*（raw data），而是使用原始数据计算得到的。

例如，如果成功生产，`prglngth` 的值就与原始变量 `wksgest`（妊娠周数）相等；否则，`prglngth` 的值估算为 `mosgest * 4.33`（妊娠月数乘以一个月的平均周数）。

重编码通常都基于一定的逻辑，这种逻辑用于检查数据的一致性和准确性。一般情况下，如果数据中存在重编码，我们就直接使用，除非有特殊的原因需要自己处理原始数据。

## 1.6 数据变换

导入调查数据时，经常需要检查数据中是否存在错误，处理特殊值，将数据转换为不同的格式并进行计算。这些操作都称为*数据清洗*（data cleaning）。

nsfg.py 包含一个 `CleanFemPreg` 函数，用于清洗计划使用的变量。

```
def CleanFemPreg(df):
```

```
df.agepreg /= 100.0

na_vals = [97, 98, 99]
df.birthwgt_lb.replace(na_vals, np.nan, inplace=True)
df.birthwgt_oz.replace(na_vals, np.nan, inplace=True)

df['totalwgt_lb'] = df.birthwgt_lb + df.birthwgt_oz / 16.0
```

`agepreg` 包含母亲在妊娠结束时的年龄。在数据文件中，`agepreg` 是以百分之一年为单位的整数值。因此 `CleanFemPreg` 的第一行将每个 `agepreg` 除以 100，从而获得以年为单位的浮点数值。

`birthwgt_lb` 和 `birthwgt_oz` 包含成功生产时的新生儿体重，分别是磅和盎司的部分。这两个变量还使用几个特殊的代码。

```
97 NOT ASCERTAINED
98 REFUSED
99 DON'T KNOW
```

用数字编码特殊值是一种危险的做法，因为如果没有进行正确的处理，这些数字可能产生虚假结果，例如，99 磅重的新生儿。`replace` 方法可以将这些值替换为 `np.nan`，这是一个特殊的浮点数值，表示“不是数字”。`replace` 方法使用 `inplace` 标识，说明直接修改现有的 Series 对象，而不是创建新对象。

IEEE 浮点数表示法标准中规定，在任何算术运算中，如果有参数为 `nan`，结果都返回 `nan`。

```
>>> import numpy as np
>>> np.nan / 100.0
nan
```

因此使用 `nan` 进行计算会得到正确的结果，而且大部分的 pandas 函数都能恰当地处理 `nan`。但我们经常需要处理数据缺失的问题。

`CleanFemPreg` 函数的最后一行生成一个新列 `totalwgt_lb`，将磅和盎司值结合在一起，得到一个以磅为单位的值。

需要注意的是，向 DataFrame 添加新列时，必须使用如下字典语法：

```
# 正确
df['totalwgt_lb'] = df.birthwgt_lb + df.birthwgt_oz / 16.0
```

而不是使用点标记：

```
# 错误!
df.totalwgt_lb = df.birthwgt_lb + df.birthwgt_oz / 16.0
```

使用点标记的写法会给 DataFrame 对象添加一个新属性，而不是创建一个新列。

## 1.7　数据验证

当数据从一个软件环境导出，再导入另一个环境时，可能会产生错误。如果不熟悉新数据集，可能会对数据进行不正确的解释，或者引入其他的误解。如果能抽出一些时间进行数据验证，就可以节省后续可能花费的时间，避免可能出现的错误。

验证数据的一种方法是计算基本的统计量，并与已发布的结果进行比较。例如，全国家庭增长调查的代码本为每个变量提供了概要表。outcome 变量对每个妊娠结果进行了编码，其概要表如下：

```
value label           Total
1 LIVE BIRTH           9148
2 INDUCED ABORTION     1862
3 STILLBIRTH            120
4 MISCARRIAGE          1921
5 ECTOPIC PREGNANCY     190
6 CURRENT PREGNANCY     352
```

Series 类提供了一个 value_counts 方法，可用于计算每个值出现的次数。如果得到 DataFrame 中的 outcome Series，我们可以使用 value_counts 方法，将结果与已发布的数据进行比较。

```
>>> df.outcome.value_counts().sort_index()
1    9148
2    1862
3     120
4    1921
5     190
6     352
```

value_counts 返回的结果是一个 Series 对象。sort_index 方法将 Series 对象按索引排序，使结果按序显示。

我们将得到的结果与官方发布的表格进行对比，outcome 变量的值似乎没有问题。类似地，已发布的关于 birthwgt_lb 的概要表如下：

```
value label                   Total
. INAPPLICABLE              4449
0-5 UNDER 6 POUNDS            1125
6 6 POUNDS                  2223
7 7 POUNDS                  3049
8 8 POUNDS                  1889
9-95 9 POUNDS OR MORE          799
```

birthwgt_lb 的 value_counts 结果如下：

```
>>> df.birthwgt_lb.value_counts(sort=False)
0        8
```

```
1       40
2       53
3       98
4      229
5      697
6     2223
7     3049
8     1889
9      623
10     132
11      26
12      10
13       3
14       3
15       1
51       1
```

数值 6、7、8 的出现次数是正确的。如果计算出 0~5 和 9~95 的次数，结果也是正确的。但是，如果再看仔细些，你会发现有一个数值肯定是错的——一个 51 磅的新生儿！

为了处理这个错误，可以在 `CleanFemPreg` 中加入一行代码。

```
df.birthwgt_lb[df.birthwgt_lb > 20] = np.nan
```

这行代码将非法值替换为 `np.nan`。方括号中的表达式产生一个 `bool` 类型的 Series 对象，值为 True 表示满足该条件。当一个布尔 Series 用作索引时，它只选择满足该条件的元素。

## 1.8　解释数据

要想有效使用数据，就必须同时在两个层面上思考问题：统计学层面和上下文层面。

例如，让我们看一看几位调查参与者的 `outcome` 序列。由于数据文件的组织方式，我们必须进行一些处理才能得到每位调查参与者的妊娠数据。以下函数实现了我们需要的处理：

```
def MakePregMap(df):
    d = defaultdict(list)
    for index, caseid in df.caseid.iteritems():
        d[caseid].append(index)
    return d
```

`df` 是包含妊娠数据的 DataFrame 对象。`iteritems` 方法遍历所有妊娠记录的索引（行号）和 `caseid`。

`d` 是将每个 `caseID` 映射到一列索引的字典。如果你不熟悉 `defaultdict`，可以到 Python 的 `collections` 模块中查看其定义。使用 `d`，我们可以查找一位调查参与者，获得其妊娠数据的索引。

下面的示例就查找了一位调查参与者，并打印出其妊娠结果列表：

```
>>> caseid = 10229
>>> indices = preg_map[caseid]
>>> df.outcome[indices].values
[4 4 4 4 4 4 1]
```

`indices` 是调查参与者 `10229` 的妊娠记录索引列表。

以这个列表为索引可以访问 `df.outcome` 中指定的行，获得一个 Series。上面的示例没有打印整个 Series 对象，而是选择输出 `values` 属性，这个属性是一个 NumPy 数组。

输出结果中的代码 `1` 表示成功分娩。代码 `4` 表示流产，即自发终止的妊娠，终止原因通常未知。

从统计学上看，这位调查参与者并无异常。流产并不少见，其他一些调查参与者的流产次数相同或者更多。

但是考虑到上下文，这个数据说明一位妇女怀孕 6 次，每次都以流产告终。她第 7 次也是最近一次怀孕成功产下了孩子。如果我们抱着同情心看待这些数据，就很容易被数据背后的故事感动。

全国家庭增长调查数据集中的每一条记录都代表一位参与者，这些参与者诚实地回答了很多非常私密而且难以回答的问题。我们可以使用这些数据解答与家庭生活、生育和健康相关的统计学问题。同时，我们有义务思及这些数据所代表的参与者，对他们心存敬意和感谢。

## 1.9 练习

- 练习 1.1

你下载的代码中应该有一个名为 chap01ex.ipynb 的文件，这是一个 IPython 记事本。你可以用如下命令从命令行启动 IPython 记事本：

```
$ ipython notebook &
```

如果系统安装了 IPython，会启动一个在后台运行的服务器，并打开一个浏览器查看记事本。如果你不熟悉 IPython，我建议你从 IPython 网站（http://ipython.org/ipython-doc/stable/notebook/notebook.html）开始学习。

你可以添加一个命令行选项，使图片在“行内”（即在记事本中）显示，而非弹出窗口：

```
$ ipython notebook --pylab=inline &
```

打开 chap01ex.ipynb。记事本中一些单元已经填好了代码，可以直接执行。其他单元列出了你应该尝试的练习。

本练习的参考答案在 chap01soln.ipynb 中。

- 练习 1.2

创建一个名为 chp01ex.py 的文件，编写代码，读取参与者文件 2001FemResp.dat.gz。你可以复制 nsfg.py 文件并对其进行修改。

变量 `pregnum` 是一个重编码，用于说明每位调查参与者有过多少次妊娠经历。打印这个变量中不同值的出现次数，将结果与全国家庭增长调查代码本中发布的结果进行比较。

你也可以将每位调查参与者的 `pregnum` 值与妊娠文件中的记录数进行比较，对调查参与者文件和妊娠文件进行交叉验证。

你可以使用 `nsfg.MakePregMap` 生成一个字典，将每个 `caseid` 映射到妊娠 DataFrame 的索引列表。

本练习的参考答案在 chp01soln.py 中。

- 练习 1.3

学习统计学的最好方法是使用一个你感兴趣的项目。你想研究“第一胎是否都会晚出生”这样的问题吗？

请思考一些你个人感兴趣的问题，可以是传统观点、争议话题或影响政局的问题，看是否可以构想出一个能以统计调查进行验证的问题。

寻找能帮助你回答这个问题的数据。公共研究的数据经常可以免费获取，因此政府网站是很好的数据来源，如 http://www.data.gov/ 和 http://www.science.gov/。如果想获得英国的数据，可以访问 http://data.gov.uk/。

我个人最喜爱的两个数据集是 General Social Survey（http://www3.norc.org/gss+website/）和 European Social Survey（http://www.europeansocialsurvey.org）。

如果有人看似已经解答了你的问题，那么仔细检查该回答是否合理。数据和分析中可能存在的缺陷都会使结论不可靠。如果发现别人的解答存在问题，你可以对同样的数据进行不同的分析，或者寻找更好的数据来源。

如果有一篇论文解答了你的问题，那么你应该能够获得论文使用的原始数据。很多论文作者会把数据放在网上供大家使用，但如果涉及敏感信息，你可能需要向作者写信索要，提供你计划如何使用这些数据的信息，或者同意某些使用条款。坚持就是胜利！

## 1.10 术语

- 轶事证据（anecdotal evidence）
  随意收集，而非通过精心设计的研究获得的证据，通常是个人证据。

- 总体（population）
  在研究中，我们感兴趣的群组。“总体”经常指一组人，但这个词也可以用于其他对象。

- 横截面研究（cross-sectional study）
  收集一个总体在某个特定时间点的数据的研究。

- 周期（cycle）
  在重复进行的横截面研究中，每次研究称为一个周期。

- 纵向研究（longtitudinal study）
  在一段时间内跟踪一个总体的研究，从同一个群体重复收集数据。

- 记录（record）
  在数据集中，关于单个人或其他对象的信息集合。

- 调查参与者（respondent）
  参与调查的人。

- 样本（sample）
  总体中用于数据收集的一个子集。

- 有代表性（representative）
  如果总体中的每个成员被选入样本的机会都均等，那么这个样本就是有代表性的。

- 过度抽样（oversampling）
  一种通过增加一个子总体的样本数来避免因样本规模过小产生错误的技术。

- 原始数据（raw data）
  没有经过或只经过少许检查、计算或解释，直接收集和记录的值。

- 重编码（recode）
  通过计算和应用于原始数据的其他逻辑生成的值。

- 数据清洗（data cleaning）
  数据处理过程，包括数据验证、错误检查，以及数据类型和表示的转换等。

# 第2章 分布

描述变量的最佳方法之一是列出该变量在数据集中的值，以及每个值出现的次数。这种描述称为该变量的*分布*（distribution）。

分布最常用的呈现方法是*直方图*（histogram），即展示每个值的*频数*（frequency）的图形。在这里，“频数”指一个值出现的次数。

使用 Python 计算频数的一种有效方法是使用字典。假设有一个值序列 `t`。

```
hist = {}
for x in t:
    hist[x] = hist.get(x, 0) + 1
```

执行这段代码，可以得到一个将值映射到频数的字典。此外，也可以使用 `collections` 模块中定义的 `Counter` 类。

```
from collections import Counter
counter = Counter(t)
```

结果是一个 `Counter` 对象，是字典类的子类。

还有一个方法是使用前一章介绍的 pandas 方法 `value_counts`。但是我为本书创建了一个类 Hist，用来表示直方图，并提供操作方法。

## 2.1　表示直方图

Hist 的构造函数参数可以是序列、字典、pandas 的 Series 对象，或者另一个 Hist 对象。你可以使用如下代码初始化一个 Hist 对象：

```
>>> import thinkstats2
>>> hist = thinkstats2.Hist([1, 2, 2, 3, 5])
>>> hist
Hist({1: 1, 2: 2, 3: 1, 5: 1})
```

Hist 对象提供 `Freq` 方法，其参数为一个值，返回结果是这个值的频数。

```
>>> hist.Freq(2)
2
```

方括号操作符也是一样。

```
>>> hist[2]
2
```

如果传入的参数值在 Hist 中不存在，频数就是 0。

```
>>> hist.Freq(4)
0
```

`Values` 方法返回 Hist 对象中值的未排序列表。

```
>>> hist.Values()
[1, 5, 3, 2]
```

如果需要按序遍历 Hist 中的值，可以使用内建函数 `sorted`。

```
for val in sorted(hist.Values()):
    print(val, hist.Freq(val))
```

也可以使用 `Items` 遍历值 – 频数对：

```
for val, freq in hist.Items():
    print(val, freq)
```

## 2.2　绘制直方图

我为本书编写了一个 `thinkplot.py` 模块，可以提供各种函数用于绘制 Hist 及文件 thinkstats2.py 中定义的其他对象。这个模块基于 `matplotlib` 包中的 `pyplot`。前言中介绍了如何安装 `matplotlib` 包。

要使用 `thinkplot` 绘制 `hist` 对象，可以尝试如下代码：

```
>>> import thinkplot
>>> thinkplot.Hist(hist)
>>> thinkplot.Show(xlabel='value', ylabel='frequency')
```

`thinkplot` 的文档位于 http://greenteapress.com/thinkstats2/thinkplot.html。

## 2.3 全国家庭增长调查中的变量

让我们回到全国家庭增长调查的数据。本章的代码在 first.py 中。前言中介绍了如何下载和使用本书代码。

刚开始使用一个新数据集时，我建议你逐个探索计划用到的变量，使用直方图就是一个很好的方法。

在 1.6 节中，我们将 `agepreg` 变量的单位从百分之一年转换为年，并将 `birthwgt_lb` 和 `birthwgt_oz` 结合生成一个数值 `totalwgt_lb`。在这一节中，我将使用这些变量展示直方图的一些特点。

首先，读入数据，选取成功生产的记录。

```
preg = nsfg.ReadFemPreg()
live = preg[preg.outcome == 1]
```

方括号中的表达式是一个布尔型 Series，从 DataFrame 中选取满足条件的行，返回一个新的 DataFrame。接下来，要为成功生产记录的 `birthwgt_lb` 生成并绘制直方图。

```
hist = thinkstats2.Hist(live.birthwgt_lb, label='birthwgt_lb')
thinkplot.Hist(hist)
thinkplot.Show(xlabel='pounds', ylabel='frequency')
```

如果 Hist 方法的参数是一个 pandas Series 对象，对象中的 `nan` 值都将去除。`label` 是图例字符串。

图 2-1 展示了前一段代码的结果。结果中出现最多的值为 7 磅，这个值称为*众数*（mode）。这个分布大致为钟形。钟形是*正态*（normal）分布，即*高斯*（Gaussian）分布的形状。但是，结果的分布是不对称的，相较右方，*尾端*（tail）向左延伸更长，这一点与正态分布不符。

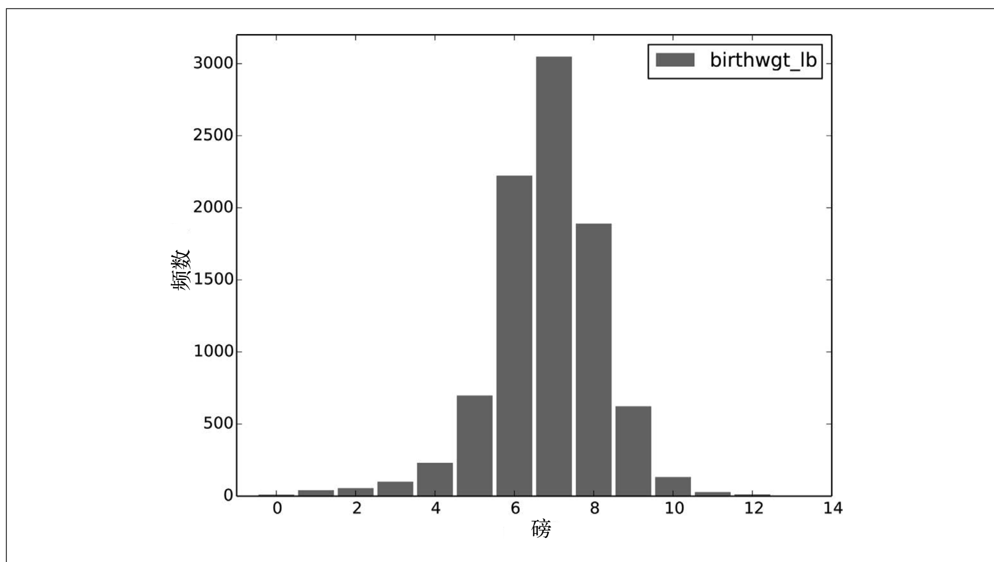



图 2-1：新生儿体重的磅值直方图

图 2-2 展示了变量 `birthwgt_oz` 的直方图，该变量表示新生儿体重的盎司值。理论上，我们预期这个分布是均匀（uniform）分布，即所有值都具有相同的频数。实际上，0 的频数最高，1 和 15 频数最低。这可能是因为调查参与者将接近整数的体重值进行了四舍五入。

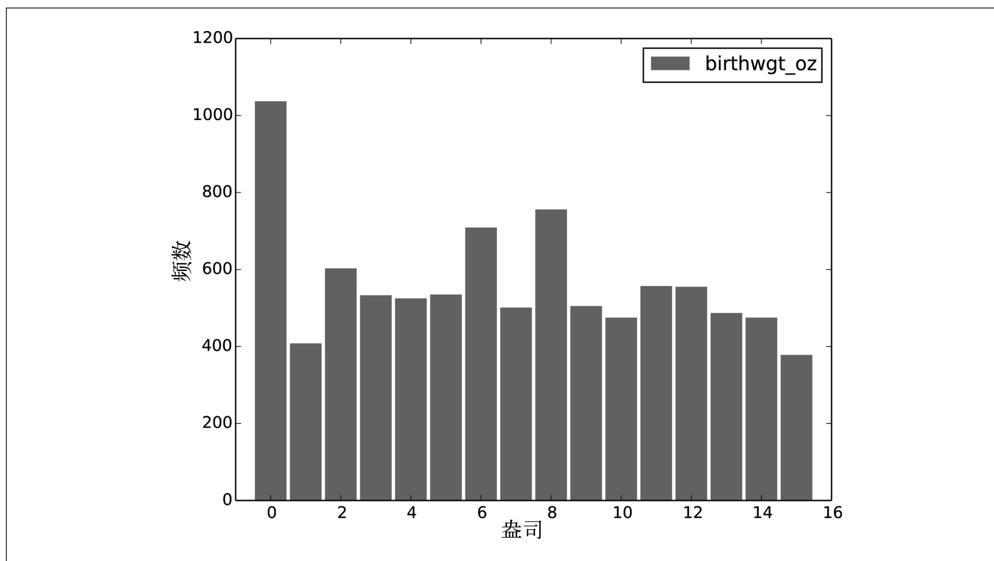



图 2-2：新生儿体重的盎司值直方图

图 2-3 展示了变量 `agepreg` 的直方图，该变量表示产妇在妊娠结束时的年龄。这一分布的

众数为 21 岁，分布形状大致为钟形，但是尾端向右延伸较长。大部分产妇年龄为 20 多岁，较少为 30 多岁。

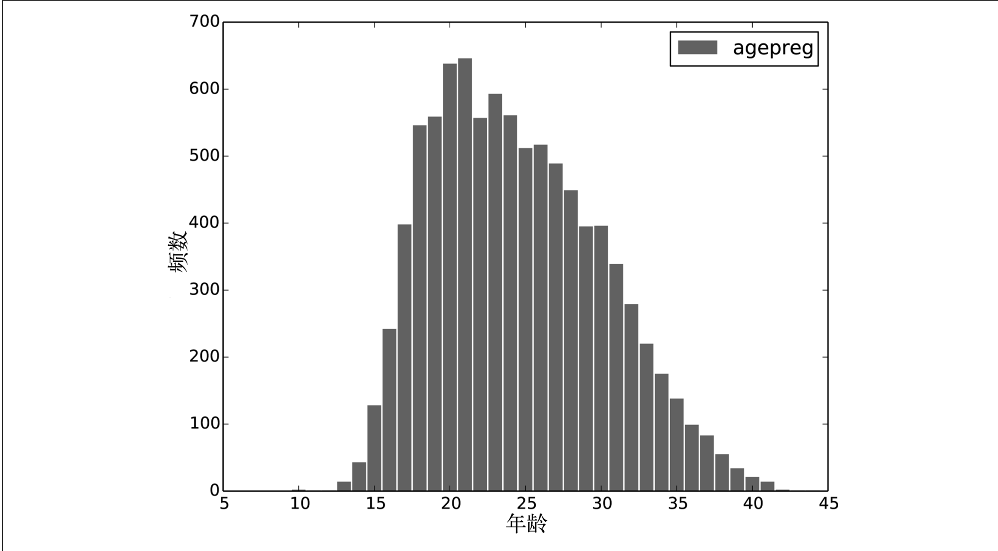



图 2-3：产妇在妊娠结束时的年龄直方图

图 2-4 展示了变量 `preglngth` 的直方图，该变量表示妊娠周数。图中最常出现的值为 39 周。这一分布的左尾比右尾更长。妊娠期少于 39 周的并不少见，但是很少有超过 43 周的。如果妊娠期超过 43 周，医生通常会进行干预。

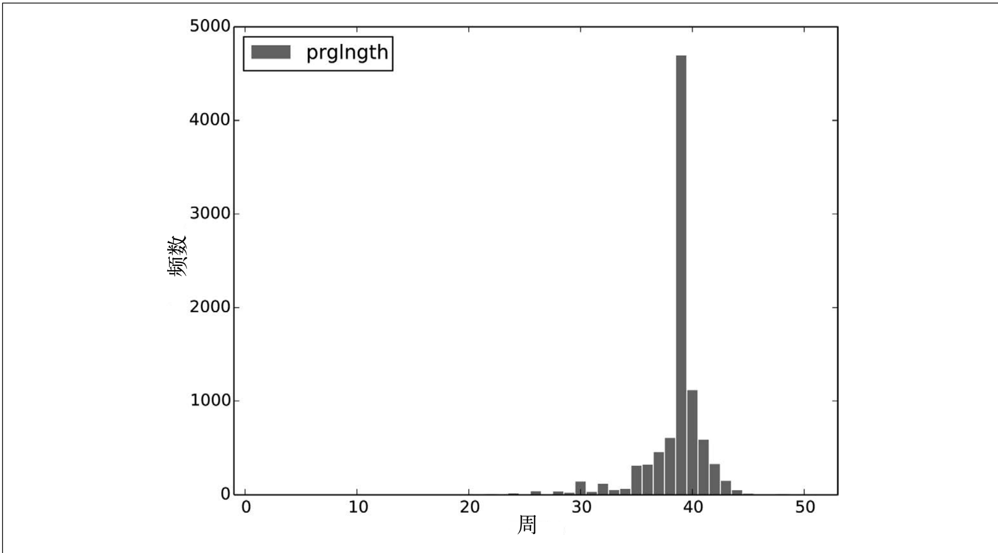



图 2-4：妊娠周数直方图

## 2.4 离群值

通过观察直方图，我们很容易发现最常出现的值，并能判断分布的形状，但不一定能看到很少出现的值。

在进一步探索数据之前，我们最好检查一下*离群值*（outlier）。离群值是极端值，可能是测量和记录中出现的错误，也可能是偶然事件的准确汇报。

Hist 对象提供 `Largest` 和 `Smallest` 方法，这两个方法的参数都是整数 `n`，分别返回直方图中 `n` 个最大和最小的值。

```
for weeks, freq in hist.Smallest(10):
    print(weeks, freq)
```

在成功生产记录的妊娠期列表中，最小的 10 个值为 `[0, 4, 9, 13, 17, 18, 19, 20, 21, 22]`。10 周以下的值肯定是错误的，很可能是结果数据没有进行正确编码；大于 30 周的数据很可能是正确的；10~30 周的数据就很难判断了，有些可能是错误的，但有些可能的确是早产儿。

另一端，最大的 10 个值为：

```
weeks  count
43     148
44     46
45     10
46     1
47     1
48     7
50     2
```

当妊娠期超过 42 周时，大部分医生会建议催产，因此大于 42 周的数据是令人惊讶的。从医学角度看，50 周几乎是不可能的。

处理离群值的最佳方法依赖于“领域知识”，即有关数据来源和意义的信息，另外还取决于你打算对数据进行何种分析。

在这个示例中，我们要解答的问题是第一胎是否会早产（或超过预产期）。当人们提出这个问题时，他们感兴趣的通常是足月妊娠，因此在这次分析中我将关注超过 27 周的妊娠记录。

## 2.5 第一胎

现在，我们可以比较第一胎和其他胎的妊娠周数分布了。我将成功生产的 DataFrame 按照 `birthord` 值进行划分，并计算其直方图。

```
firsts = live[live.birthord == 1]
others = live[live.birthord != 1]

first_hist = thinkstats2.Hist(firsts.prglngth)
other_hist = thinkstats2.Hist(others.prglngth)
```

然后在同一坐标轴上绘制这两个直方图。

```
width = 0.45
thinkplot.PrePlot(2)
thinkplot.Hist(first_hist, align='right', width=width)
thinkplot.Hist(other_hist, align='left', width=width)
thinkplot.Show(xlabel='weeks', ylabel='frequency')
```

`thinkplot.PrePlot` 的参数是计划绘制的直方图数量，`thinkplot` 使用这一信息选择适当的绘制颜色。

`thinkplot.Hist` 通常使用 `align='center'`，从而使每个柱形以其值为中心进行显示。在这个图形中，我对两个直方图分别设置了 `align='right'` 和 `'align='left'`，使其柱形分别在值的两边显示。

通过设置 `width=0.45` 让两个直方图的柱形总宽度为 0.9，每对柱形之间留有一些空隙。

最后，调整轴设置，只显示 27~46 周的数据。图 2-5 展示了绘制结果。

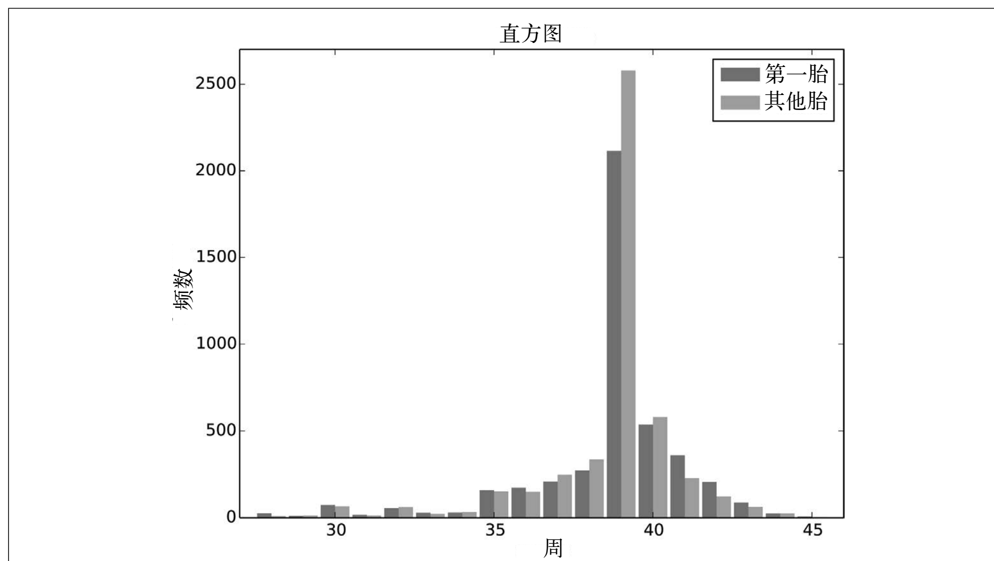



图 2-5：妊娠周数直方图

直方图清晰展示了最常出现的变量值，因此非常有用。但是，直方图并不是比较两个变量分布的最佳选择。在这个示例中，“第一胎”的数量比“其他胎”少，因此图中展示的一

些明显差异是由样本规模导致的。在下一章中，我们将使用概率质量函数解决这个问题。

## 2.6 分布概述

直方图是一个样本分布的完整描述，也就是说，有了直方图，我们可以重构样本中的值（但无法重构其顺序）。

如果一个分布的细节很重要，我们可能需要展示其直方图。但是，我们经常只需使用几个描述性的统计量，对变量分布进行一个概述。

我们可能需要描述的变量分布特征有如下这些。

- *集中趋势*
  变量值是否聚集在某个值的附近？
- *众数*
  是否有多个聚集点？
- *展布*
  变量的变化性如何？
- *尾部*
  当值偏离众数时，其概率降低多快？
- *离群值*
  是否有远离众数的极端值？

*汇总统计量*（summary statistic）就是为回答以上这些问题而设计的。目前最常用的汇总统计量是*均值*（mean），用于描述分布的*集中趋势*（central tendency）。

如果有一个样本，其中包含 $n$ 个值 $x_i$，均值$\bar{x}$就是所有值的总和除以值的个数，即：

$$\bar{x} = \frac{1}{n}\sum_i x_i$$

“均值”和“平均数”这两个词有时可以互换使用，但有一个区别：

- 一个样本的“均值”是使用上面的公式计算得到的汇总统计量；
- “平均数”是描述集中趋势的汇总统计量之一。

有时，均值可以很好地描述一组值。例如：苹果的大小都差不多（至少超市卖的苹果如此）。因此，如果我买了 6 个苹果，总重量为 3 磅，那么每个苹果约重半磅就是合理的说法。

但是南瓜个体差异比较大。假设我在院子里种了好几种南瓜，某天收获了 3 个做装饰的南瓜，每个 1 磅重；2 个做派的南瓜，每个 3 磅重；还有 1 个 Atlantic Giant 南瓜，重达 591 磅。我收获的南瓜样本均值为 100 磅。但是如果我说"我院子里普通大小的南瓜重 100 磅"，那就是在误导你。这个例子中不存在具有代表性的南瓜，因此平均值没有意义。

## 2.7　方差

如果我们不能用一个值来概括南瓜的重量，那么使用两个值会好一些：均值和*方差*（variance）。

方差是用于描述一个分布的变化性或者*展布*（spread）的汇总统计量。计算方差的公式为：

$$S^2 = \frac{1}{n}\sum_i (x_i - \bar{x})^2$$

公式中的 $x_i - \bar{x}$ 称为"离差"，因此方差就是离差平方的均值。方差的平方根 $S$ 是*标准差*（standard deviation）。

如果你以前学过相关知识，可能见过以 $n-1$ 而不是以 $n$ 为分母的方差公式。这种统计量用于使用一个样本对总体方差进行估计。我们将在第 8 章对此进行讨论。

Pandas 数据结构提供计算均值、方差和标准差的方法。

```
mean = live.prglngth.mean()
var = live.prglngth.var()
std = live.prglngth.std()
```

对于所有成功生产的妊娠数据，妊娠期的均值为 38.6 周，标准差为 2.7 周，也就是说，我们认为 2~3 周的偏差值是正常的。

妊娠期的方差为 7.3，这个值很难解释，尤其是方差的单位是周的平方，或"平方周"。方差在某些计算中有用，却不是一个很好的汇总统计量。

## 2.8　效应量

*效应量*（effect size）是用于描述效应大小的汇总统计量。例如，要描述两个群组之间的差异，一个显而易见的选择是使用均值的差值。

第一胎妊娠期的均值是 38.601，非第一胎的妊娠期均值是 38.523。二者差值为 0.078 周（即 13 小时），将这个值除以平均的妊娠周数，得出差异约为 0.2%。

如果我们假设这个估算是准确的，那么这个差值不具有实际意义。实际上，如果没有对大

量妊娠数据进行研究，根本不可能有人会注意到这一差异。

另一个描述效应量的方法是将群组之间的差值与群组内的变化性进行比较。Cohen's $d$ 就是这样一个统计量，其定义如下：

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s}$$

其中 $\bar{x}_1$ 和 $\bar{x}_2$ 是各群组的均值，$s$ 是“合并标准差”（pooled standard deviation）。计算 Cohen's $d$ 的 Python 代码如下：

```
def CohenEffectSize(group1, group2):
    diff = group1.mean() - group2.mean()

    var1 = group1.var()
    var2 = group2.var()
    n1, n2 = len(group1), len(group2)

    pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
    d = diff / math.sqrt(pooled_var)
    return d
```

在我们的示例中，两组样本的均值差为 0.029 个标准差，这个值很小。对比一下，男性和女性的身高差约为 1.7 个标准差（参见 https://en.wikipedia.org/wiki/Effect_size）。

## 2.9 报告结果

对于第一胎和其他胎的妊娠期长度差异（如果存在差异的话），我们已经讨论了好几种方法来进行描述。我们应该如何报告这些结果呢？

答案取决于提出问题的人。科学家可能对任何（真实的）效应都感兴趣，无论该效应多么微小；医生可能只关心那些临床显著（clinically significant）的效应，即影响治疗诊断的效应；孕妇则可能对与自己有关的结果感兴趣，例如早于或晚于预产期生产的可能性。

你的目标也会影响结果的报告。如果你试图证明一个效应的重要性，那么可能会选择使用强调差异的汇总统计量；如果你要安慰一位病人，那么可能会选择使该差异更具有实际意义的统计量。

当然，你的选择还应该遵守职业道德。使用数据进行说明是合理的，你应当设计统计图形和报告清晰表达自己的观点。但是，你也应该尽量诚实地报告结果，并承认结论的不确定性和研究的局限性。

## 2.10　练习

- 练习 2.1

基于本章结果，总结一下关于第一胎是否晚于预产期出生这一问题，你都学到了什么。

如果要在晚间新闻上发布一条消息，你会使用哪些汇总统计量？如果要安慰一位焦虑的病人，你又会选择哪些汇总统计量？

最后，假设你是 *The Straight Dope*（http://straightdope.com）的作者 Cecil Adams，任务是回答“第一胎是否会晚于预产期出生”。请使用本章结果撰写一段文字，清晰、准确且诚实地回答这个问题。

- 练习 2.2

在你下载的代码库中，有一个名为 chap02ex.ipynb 的文件，请打开这个文件。一些单元已经填好代码，请执行。其他单元给出了练习指令，请按照指令练习并填写答案。

此练习的参考答案在 chap02soln.ipynb 中。

对于接下来的练习，请创建文件 chap02ex.py。你可以在 chap02soln.py 中找到参考答案。

- 练习 2.3

一个分布的众数是其中最常出现的值（参见 http://wikipedia.org/wiki/Mode_(statistics)）。请编写一个函数 `Mode`，使用 Hist 参数，返回最常出现的值。

要进一步挑战自己，请编写一个函数 `AllModes`，返回一个值 – 频数对列表，列表按频数降序排列。

- 练习 2.4

请使用变量 `totalwgt_lb`，研究第一胎是否比其他新生儿体重轻 / 重。请计算 Cohen’s $d$，对群组之间的差异进行量化。将得到的结果与妊娠期差异的结果对比，有何不同？

## 2.11　术语

- 分布（distribution）
  样本中的值以及每个值出现的频数。

- 直方图（histogram）
  从值到频数的映射，或者展示这一映射的图形。

- 频数（frequency）
  一个值在样本中出现的次数。

- 众数（mode）
  一个样本中最常出现的值，或者最常出现的值之一。

- 正态分布（normal distribution）
  钟形的理想化分布，也称为高斯分布。

- 均匀分布（uniform distribution）
  所有值具有相同频数的分布。

- 尾部（tail）
  一个分布中最高端和最低端的部分。

- 集中趋势（central tendency）
  样本或总体的一个特征。直观上，这一特征是一个平均值或典型值。

- 离群值（outlier）
  远离集中趋势的值。

- 展布（spread）
  对值在分布中扩展规模的度量。

- 汇总统计量（summary statistic）
  对分布的某些方面（如集中趋势或展布）进行量化的统计量。

- 方差（variance）
  一种汇总统计量，常用于量化展布。

- 标准差（standard deviation）
  方差的平方根，也用于量化展布。

- 效应量（effect size）
  一种汇总统计量，用于量化一个效应的大小，如群组之间的差异。

- 临床显著（clinically significant）
  在实践中有意义的结果，如群组之间的差异。

# 第3章

# 概率质量函数

这一章的代码在 probability.py 中。前言介绍了如何下载和使用本书代码。

## 3.1 概率质量函数

除了直方图，另一种可以表示分布的方法是*概率质量函数*（probability mass function，PMF）。概率质量函数将每个值映射到其概率。*概率*（probability）是频数的分数表示，样本量为 `n`。要从频数计算出概率，我们将频数除以 `n`，这一过程称为*正态化*（normalization）。

给定一个 Hist 对象，我们可以创建一个字典，将每个值映射到对应的概率。

```
n = hist.Total()
d = {}
for x, freq in hist.Items():
    d[x] = freq / n
```

也可以使用 `thinkstats2` 提供的 Pmf 类。和 Hist 类一样，Pmf 的构造函数参数可以是列表、pandas Series、字典、Hist 对象或另一个 Pmf 对象。下面是一个使用简单列表创建 Pmf 的例子。

```
>>> import thinkstats2
>>> pmf = thinkstats2.Pmf([1, 2, 2, 3, 5])
>>> pmf
Pmf({1: 0.2, 2: 0.4, 3: 0.2, 5: 0.2})
```

Pmf 将各个值的频数进行了正态化，因此概率总和为 1。

Pmf 和 Hist 对象在很多方面都很相似。实际上，这两个类的很多方法都继承自同一个父类。例如，Pmf 和 Hist 都有 `Values` 和 `Items` 方法，用法也相同。二者最大的区别是，Hist 将值映射到整型的计数值，而 Pmf 将值映射到浮点型的概率值。

使用 `Prob` 方法可以查找一个值对应的概率值。

```
>>> pmf.Prob(2)
0.4
```

使用括号写法也可以实现同样的功能。

```
>>> pmf[2]
0.4
```

你可以修改一个现有的 Pmf 对象，增加一个值的概率。

```
>>> pmf.Incr(2, 0.2)
>>> pmf.Prob(2)
0.6
```

或者将其概率值乘以一个倍数。

```
>>> pmf.Mult(2, 0.5)
>>> pmf.Prob(2)
0.3
```

如果你修改了一个 Pmf 对象，其结果可能不再是正态化的，即所有值的概率和不再为 1。要检查 Pmf 对象的结果是否正态化，你可以调用 `Total` 方法，返回所有值的概率和。

```
>>> pmf.Total()
0.9
```

要重新进行正态化，可以调用 `Normalize` 方法。

```
>>> pmf.Normalize()
>>> pmf.Total()
1.0
```

Pmf 对象提供了 `Copy` 方法，可用于在保持原对象不变的前提下创建和修改一个副本。

这一节中使用的记号看似有些混乱，其实也有规律可循：首字母大写的 Pmf 是类名，小写的 `pmf` 是类的一个实例，全大写的 PMF 是指概率质量函数的数学概念。

## 3.2　绘制PMF

`thinkplot` 提供两种绘制 Pmf 的方法。

- 使用 `thinkplot.Hist` 将 Pmf 绘制为条形图。当 Pmf 中值的数量较少时，条形图最为实用。

- 使用 thinkplot.Pmfs 将 Pmf 绘制为阶梯函数。如果 Pmf 中值的数量很多而且分布曲线平滑，这个方法最为适宜。这个函数也可以绘制 Hist 对象。

此外，pyplot 还提供一个 hist 函数，以一列值为参数，计算出一个直方图，并进行绘制。既然我已经使用了 Hist 对象，就不必使用 pyplot.hist 函数了。

图 3-1 展示了第一胎和其他胎的妊娠期 PMF 的条形图（左侧）和阶梯函数（右侧）。

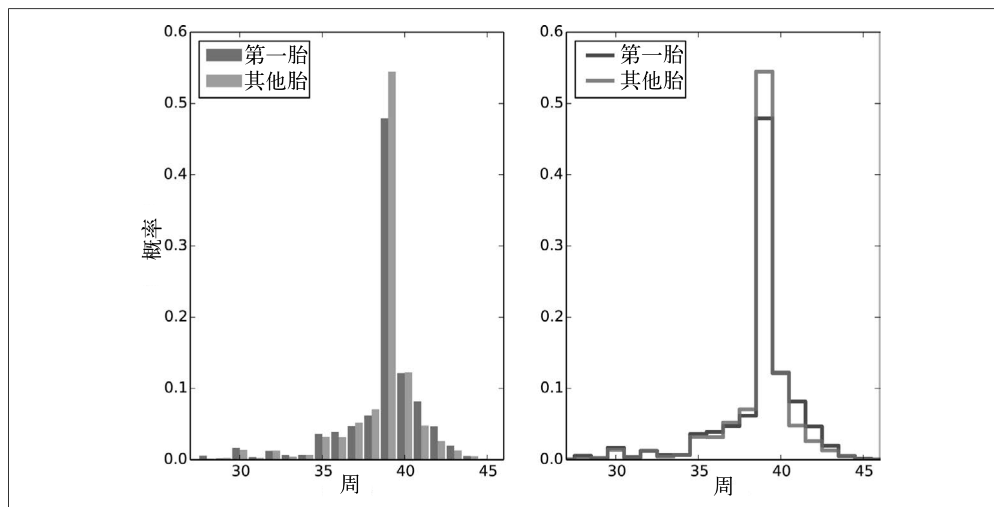



图 3-1：第一胎和其他胎的妊娠期 PMF 的条形图和阶梯函数

生成图 3-1 的代码如下：

```
thinkplot.PrePlot(2, cols=2)
thinkplot.Hist(first_pmf, align='right', width=width)
thinkplot.Hist(other_pmf, align='left', width=width)
thinkplot.Config(xlabel='weeks',
                 ylabel='probability',
                 axis=[27, 46, 0, 0.6])

thinkplot.PrePlot(2)
thinkplot.SubPlot(2)
thinkplot.Pmfs([first_pmf, other_pmf])
thinkplot.Show(xlabel='weeks',
               axis=[27, 46, 0, 0.6])
```

通过绘制 PMF 而非直方图，我们可以不受样本量差异的影响，对两个分布进行比较。从图中我们可以看到，比起其他胎，第一胎似乎较少会准时出生（39 周），而更可能会较晚出生（41 周和 42 周）。

PrePlot 通过两个可选参数 rows 和 cols 将图形显示在网格中，上述代码设置将图形显示为一行两列。第一个图形（左侧）使用前面介绍过的 thinkplot.Hist 显示 Pmf。

代码中第二次调用的 PrePlot 方法重置了色彩生成器，然后 SubPlot 切换到第二个图形（右侧），使用 thinkplot.Pmf 绘制 Pmf 对象。代码中使用了 axis 参数，确保两张图位于同样的坐标轴上。如果要比较两个图形，设置 axis 参数通常是个不错的主意。

## 3.3 绘制PMF的其他方法

在探索数据并尝试发现其中的模式和关系时，直方图和 PMF 都是非常实用的。一旦你对数据有了大致的了解，下一步最好设计一种可视化方法，尽可能清晰地展现你所发现的模式。

在全国家庭增长调查数据中，第一胎和其他数据分布的最大区别位于众数附近。因此，我们应该对图形中的这部分进行放大，并转换数据，以强调这种区别。

```
weeks = range(35, 46)
diffs = []
for week in weeks:
    p1 = first_pmf.Prob(week)
    p2 = other_pmf.Prob(week)
    diff = 100 * (p1 - p2)
    diffs.append(diff)

thinkplot.Bar(weeks, diffs)
```

在这段代码中，weeks 是周数范围，diffs 是两个 PMF 差异的百分数。图 3-2 将结果展示为条形图。这张图使数据模式更加清晰：相比其他胎，第一胎较少在第 39 周出生，在第 41 周和第 42 周出生的可能性稍大。

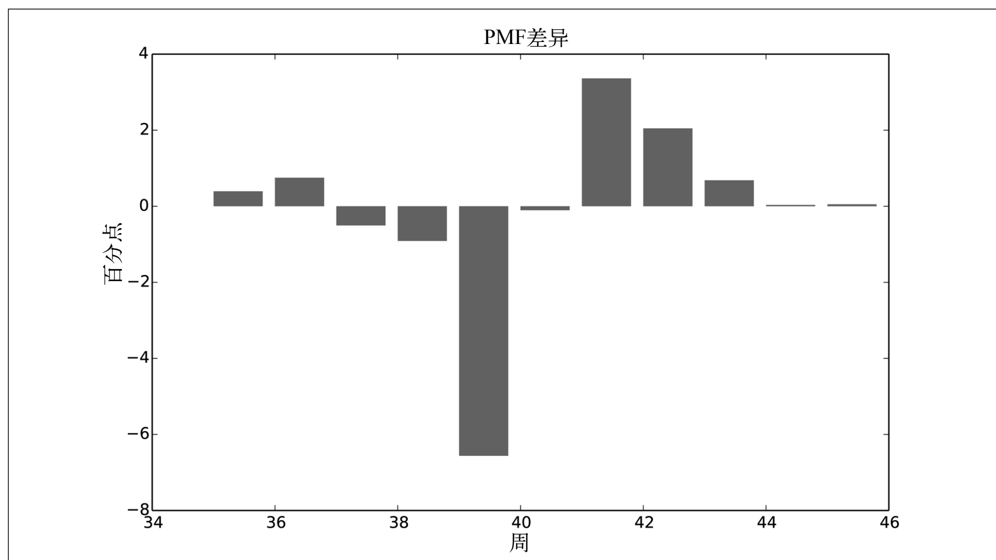



图 3-2：按周显示的 PMF 差异百分点

目前，我们应该对此结论持谨慎态度。我们使用了同一个数据集来发现一个明显差异，然后选择一种可视化方法来强调这一差异。我们不能确定这种效应是否真的存在，它也许只是随机变化的结果。本书稍后将解决这一疑问。

## 3.4 课堂规模悖论

在继续讨论之前，我要演示一种 Pmf 对象的计算。我称之为“课堂规模悖论”。

在很多美国大学和学院里，学生与教师的比率约为 10:1。但是学生们经常会惊讶地发现自己课上的平均学生数大于 10。造成这一现象的原因有两个：

- 学生通常每学期修 4 到 5 门课，但是教授经常只教 1 门或 2 门；
- 上小课的学生很少，但上大课的学生人数非常之多。

一旦我们知道了第一个原因，其效应就是显而易见的。第二个就不那么明显了。让我们来看一个例子。假设一所学院某学期开了 65 门课，选课人数分布如下：

```
 size     count
 5- 9         8
10-14         8
15-19        14
20-24         4
25-29         6
30-34        12
35-39         8
40-44         3
45-49         2
```

如果你问院长，每门课平均的选课人数是多少，他会构建一个 PMF，计算出均值，然后回答说选课人数平均为 23.7。具体实现代码如下：

```
d = { 7: 8, 12: 8, 17: 14, 22: 4,
      27: 6, 32: 12, 37: 8, 42: 3, 47: 2 }

pmf = thinkstats2.Pmf(d, label='actual')
print('mean', pmf.Mean())
```

但是如果对一组学生进行调查，询问他们的课堂上有多少学生，然后计算出均值，那么会发现选课人数的平均值比 23.7 大。具体大多少呢？

首先，计算出学生观察到的分布，其中每个课堂规模值的概率受到选课的学生人数的“影响”。

```
def BiasPmf(pmf, label):
    new_pmf = pmf.Copy(label=label)

    for x, p in pmf.Items():
```

```
            new_pmf.Mult(x, x)

        new_pmf.Normalize()
        return new_pmf
```

对每个课堂规模值 x，我们将其概率乘以 x，即观察该课堂规模的学生人数，最终能得到一个新的 Pmf，代表这个偏倚分布。

现在我们可以绘制出实际分布和观察到的分布。

```
    biased_pmf = BiasPmf(pmf, label='observed')
    thinkplot.PrePlot(2)
    thinkplot.Pmfs([pmf, biased_pmf])
    thinkplot.Show(xlabel='class size', ylabel='PMF')
```

图 3-3 展示了代码的运行结果。在偏倚分布中，小课更少，大课更多。偏倚分布的均值为 29.1，比实际的均值高出约 25%。

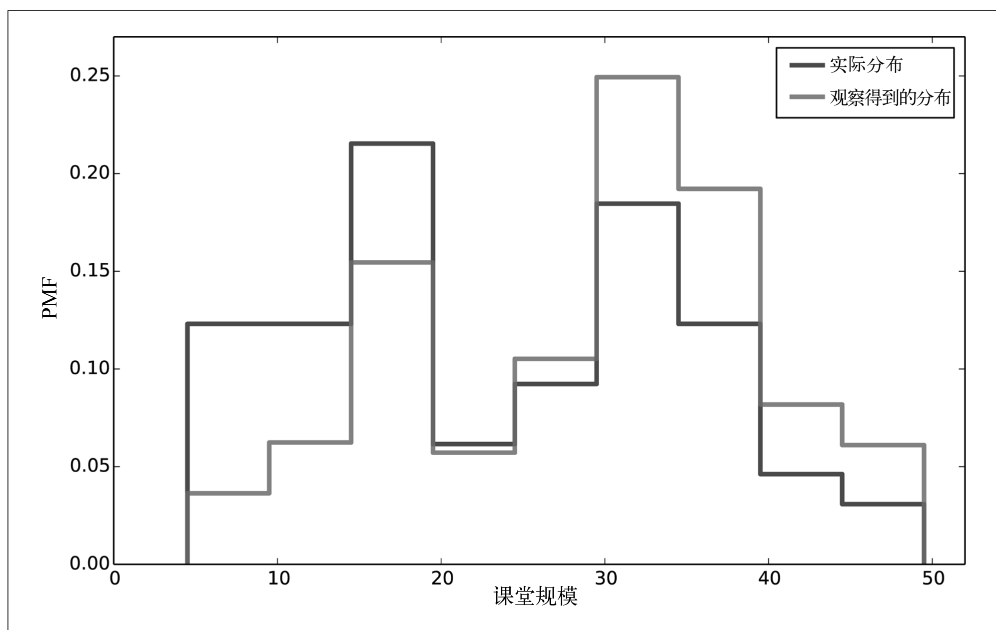



图 3-3：课堂规模的实际分布以及由学生观察得到的分布

我们也可以反向进行这一操作。假设你希望得到一所学院的课堂规模分布，但却无法从院长处得到可靠数据。你可以采用另一个办法：选择学生的一个随机样本，进行调查，询问他们的课堂上有多少学生。

因为存在我们前面讨论过的原因，所以这个调查得到的结果是偏倚的，但是你可以使用这一结果来估算实际分布。下面的函数对一个 Pmf 进行“去偏倚”操作：

```
def UnbiasPmf(pmf, label):
    new_pmf = pmf.Copy(label=label)

    for x, p in pmf.Items():
        new_pmf.Mult(x, 1.0/x)

    new_pmf.Normalize()
    return new_pmf
```

这个函数与 `BiasPmf` 很类似，唯一的区别在于这个函数将每个概率除以 `x`，而不是乘以 `x`。

## 3.5 使用DataFrame进行索引

在 1.4 节中，我们读取了一个 pandas 的 DataFrame 对象，并用这个对象选择和修改数据列。现在让我们看看如何进行数据行的选择。首先，创建一个 NumPy 随机数组，用它初始化一个 DataFrame。

```
>>> import numpy as np
>>> import pandas
>>> array = np.random.randn(4, 2)
>>> df = pandas.DataFrame(array)
>>> df
          0         1
0 -0.143510  0.616050
1 -1.489647  0.300774
2 -0.074350  0.039621
3 -1.369968  0.545897
```

默认情况下，DataFrame 的行和列都是从 0 开始计数的，但你可以提供列名。

```
>>> columns = ['A', 'B']
>>> df = pandas.DataFrame(array, columns=columns)
>>> df
          A         B
0 -0.143510  0.616050
1 -1.489647  0.300774
2 -0.074350  0.039621
3 -1.369968  0.545897
```

你也可以提供行名。行名的集合称为*索引*（index），行名则称为*标签*（label）。

```
>>> index = ['a', 'b', 'c', 'd']
>>> df = pandas.DataFrame(array, columns=columns, index=index)
>>> df
          A         B
a -0.143510  0.616050
b -1.489647  0.300774
c -0.074350  0.039621
d -1.369968  0.545897
```

正如我们在前一章看到的，你可以使用索引选择一个列，返回一个 Series 对象。

```
>>> df['A']
a   -0.143510
b   -1.489647
c   -0.074350
d   -1.369968
Name: A, dtype: float64
```

可以使用 `loc` 属性通过标签选择一行，返回一个 Series 对象。

```
>>> df.loc['a']
A   -0.14351
B    0.61605
Name: a, dtype: float64
```

如果你知道一行的整数位置，而不是它的标签，则可以使用 `iloc` 属性，它也可以返回一个 Series 对象。

```
>>> df.iloc[0]
A   -0.14351
B    0.61605
Name: a, dtype: float64
```

`loc` 属性也可以以一列标签为参数，这种情况下会返回一个 DataFrame 对象。

```
>>> indices = ['a', 'c']
>>> df.loc[indices]
         A         B
a -0.14351  0.616050
c -0.07435  0.039621
```

最后，你可以使用一个切片，通过标签选择行的范围。

```
>>> df['a':'c']
          A         B
a -0.143510  0.616050
b -1.489647  0.300774
c -0.074350  0.039621
```

使用行的整数行号也可以实现同样的操作。

```
>>> df[0:2]
          A         B
a -0.143510  0.616050
b -1.489647  0.300774
```

这两种操作都返回 DataFrame 对象，但是请注意第一个结果包含了最后一个切片，而第二个结果则不包含。

如果数据行的标签不是简单的整数，那么我建议你在代码中始终使用标签，避免使用整数的行号。

## 3.6 练习

本章练习的参考答案位于 chap03soln.ipynb 和 chap03soln.py 文件中。

- 练习 3.1

如果你对孩子进行调查，询问他们家中有几个兄弟姐妹，就会出现类似课堂规模悖论的情况：你的样本中更可能出现有很多孩子的家庭，而根本不会包括没有孩子的家庭。

请使用全国家庭增长调查的变量 `NUMKDHH` 构建家庭中未满 18 岁的成员数量的实际分布。

如果我们对孩子进行调查，询问其家中 18 岁以下成员（包括他们自己）的数量，请计算这一调查会得到的偏倚分布。

请绘制实际分布和偏倚分布，并计算这两种分布的均值。你可以在 chap03ex.ipynb 的基础上进行修改和练习。

- 练习 3.2

在 2.6 节中，我们将一个样本中的元素累加后除以 $n$，计算出样本的均值。如果给定一个 PMF，你仍然可以计算其均值，只是计算过程发生了少许变化：

$$\bar{x} = \sum_i p_i x_i$$

其中 $x_i$ 是 PMF 中的不同值，$p_i$=PMF($x_i$)。类似地，你也可以用如下公式计算方差：

$$S^2 = \sum_i p_i (x_i - \bar{x})^2$$

请编写函数 `PmfMean` 和 `PmfVar`，以 Pmf 对象为参数，分别计算其均值和方差。请检查你编写的函数结果与 Pmf 自带的 `Mean` 和 `Var` 方法结果是否一致，以验证函数实现是否正确。

- 练习 3.3

本书开头提出问题：“第一胎是否经常晚于预产期出生？”为了回答这一问题，我计算了不同群组新生儿的均值差异，但却忽略了另一种可能性：同一位母亲的第一胎和其他孩子之间可能存在差异。

要回答这个问题，请选择至少生育两胎的调查参与者，计算不同群组的差异。结果是否不同？

提示：使用 `nsfg.MakePregMap`。

- 练习 3.4

在大部分跑步比赛中，参赛者都同时出发。如果你速度很快，通常会在比赛刚开始后超越

很多参赛者，但在跑出几英里之后，周围的选手就和你速度一样了。

第一次参加长距离接力（209 英里）比赛时，我注意到一个奇怪的现象：当我超越其他选手时，我的速度通常快得多；当别的选手超越我时，他通常比我跑得快多了。

一开始，我认为参赛者的速度分布可能两极分化，有很多跑得很慢的人，也有很多跑得很快的人，但是与我速度相同的人很少。

后来我意识到自己受到了类似课堂规模效应的影响。这场比赛有两点特殊性：比赛是分段开始的，即各支队伍出发的时间不同；而且很多队伍中选手的能力各不相同。

因此，选手们分布在赛道各处，速度和位置的关系并不大。当我加入比赛时，周围的选手（几乎）是参赛选手的一个随机样本。

那么这种偏倚是怎么产生的呢？在奔跑过程中，超越别人和被别人超越的机会是与我们的速度差成比例的。我更有可能追上速度比我慢的选手，也更有可能被速度更快的选手超越。但速度相同的选手却不太可能相遇。

请编写一个函数 `ObservedPmf`，以参赛者速度实际分布的 Pmf 和一位参赛观察者的速度为参数，返回一个新的 Pmf，表示这位观察者看到的选手的速度分布。

你可以使用 relay.py 测试自己的函数。relay.py 文件读取麻省 Dedham 的 James Joyce Ramble 10 公里马拉松比赛的结果数据，并将每位参赛者的速度转换为每小时英里数。

假设你与这群选手一起以每小时 7.5 英里的速度参加一场接力赛，请计算你将观察到的速度分布。本练习的参考答案在 relay_soln.py 中。

## 3.7 术语

- 概率质量函数（probability mass function，PMF）
  将概率分布表示为从值到概率的映射。

- 概率（probability）
  将频数表示为样本量的分数。

- 正态化（normalization）
  将频数除以样本量以获得概率值的过程。

- 索引（index）
  在 pandas 的 DataFrame 中，索引是包含行标签的特殊列。

# 第4章 累积分布函数

本章代码位于 cumulative.py 中。前言介绍了如何下载和使用本书代码。

## 4.1 PMF的局限

PMF 适用于变量值数量较少的情况。但是随着值的数量增加，每个值对应的概率会变得越来越小，随机噪音的影响就会变大。

例如，我们可能对新生儿的体重分布感兴趣。在全国家庭增长调查的数据中，变量 `totalwgt_lb` 记录了新生儿的体重，单位为磅。图 4-1 展示了第一胎新生儿和其他胎变量 `totalwgt_lb` 的 PMF。

这些分布大致接近正态分布的钟形，靠近均值的值较多，两端的值较少。

但是我们很难对这个图形中的某些部分进行解释。图中有很多的“尖峰”和“低谷”，而且两个分布有明显差别。从图中我们很难分辨哪些特征是有意义的，而且也不容易看出整体模式，如哪个分布的均值更高。

为了解决上面的问题，我们可以将数据分区，即将值的的范围划分为互不重叠的区间，然后计算每个区间中值的数目。分区是很实用的方法，但选择区间大小并不容易。如果区间太大，在消除噪音的同时，也可能会消除有用的信息。

避免上述问题的另一个方法是使用*累积分布函数*（cumulative distribution function，CDF）。这一章将主要介绍 CDF，但在此之前，我们需要了解百分位数。

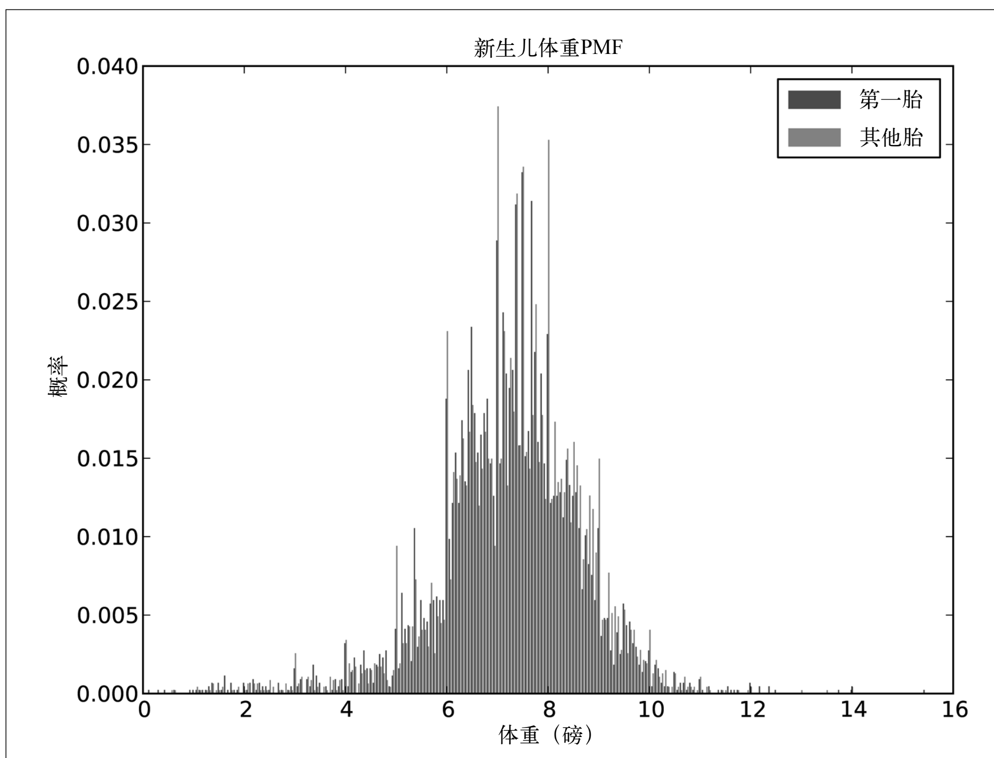



图 4-1：新生儿体重的 PMF 展示了 PMF 的局限：难以通过观察进行比较

## 4.2 百分位数

如果你参加过标准化考试，得到的结果可能是原始成绩和*百分位秩*（percentile rank）。在标准化考试成绩中，百分位秩是比你成绩低（或相同）的人的比例。因此，如果你“位于第 90 百分位”，就说明你的成绩高于或等于 90% 参加考试的人。

要计算序列 `scores` 中的一个值 `your_score` 的百分位秩，可以使用如下代码：

```
def PercentileRank(scores, your_score):
    count = 0
    for score in scores:
        if score <= your_score:
            count += 1

    percentile_rank = 100.0 * count / len(scores)
    return percentile_rank
```

举个例子，如果序列中的成绩值为 56、66、77、88 和 99，你的成绩是 88，那么你的百分位秩就是 `100 * 4 / 5`，即 80。

从一个值计算其百分位秩很容易，但反过来就有些难度了。如果给定一个百分位秩，要找出其对应的值，可以对值排序，并进行查找。

```
def Percentile(scores, percentile_rank):
    scores.sort()
    for score in scores:
        if PercentileRank(scores, score) >= percentile_rank:
            return score
```

这个计算的结果是一个*百分位数*（percentile）。例如，第 50 百分位数是百分位秩为 50 的值。在之前给出的考试成绩分布中，第 50 百分位数是 77。

`Percentile` 的实现方法性能欠佳。更好的方法是使用百分位秩计算相应的百分位数索引。

```
def Percentile2(scores, percentile_rank):
    scores.sort()
    index = percentile_rank * (len(scores)-1) / 100
    return scores[index]
```

“百分位数”和“百分位秩”的概念容易混淆，常有人用错。概括地说，`PercentileRank` 以一个值为参数，计算这个值在一组值中的百分位秩；`Percentile` 以一个百分位秩为参数，计算对应的值。

## 4.3 CDF

理解百分位数和百分位秩的概念后，我们就可以开始讨论*累积分布函数*（cumulative distribution function，CDF）了。CDF 将一个值映射到百分位秩。

CDF 是 $x$ 的函数，其中 $x$ 是可能出现在分布中的任意值。要获得某个特定值 $x$ 的 CDF($x$)，我们需要计算出小于或等于 $x$ 的值在此分布中所占的比例。

下面代码中的函数参数为一个序列 `t` 和一个值 `x`。

```
def EvalCdf(t, x):
    count = 0.0
    for value in t:
        if value <= x:
            count += 1

    prob = count / len(t)
    return prob
```

这个函数和前一节中定义的 `PercentileRank` 几乎一模一样，但是这个函数的结果为 0 到 1 的概率，而 `PercentileRank` 的结果是 0 到 100 的百分位秩。

举个例子，假设我们收集到的样本值为 `[1, 2, 2, 3, 5]`，以下是这个样本的一些 CDF 值：

CDF(0) = 0

CDF(1) = 0.2

CDF(2) = 0.6

CDF(3) = 0.8

CDF(4) = 0.8

CDF(5) = 1

我们可以为任意值 $x$ 计算 CDF，而不仅限于样本中出现的值。如果 $x$ 小于样本中的最小值，那么 CDF($x$) 为 0。如果 $x$ 大于样本中的最大值，那么 CDF($x$) 为 1。

图 4-2 展示了这个 CDF 函数。一个样本的 CDF 是一个阶梯函数。

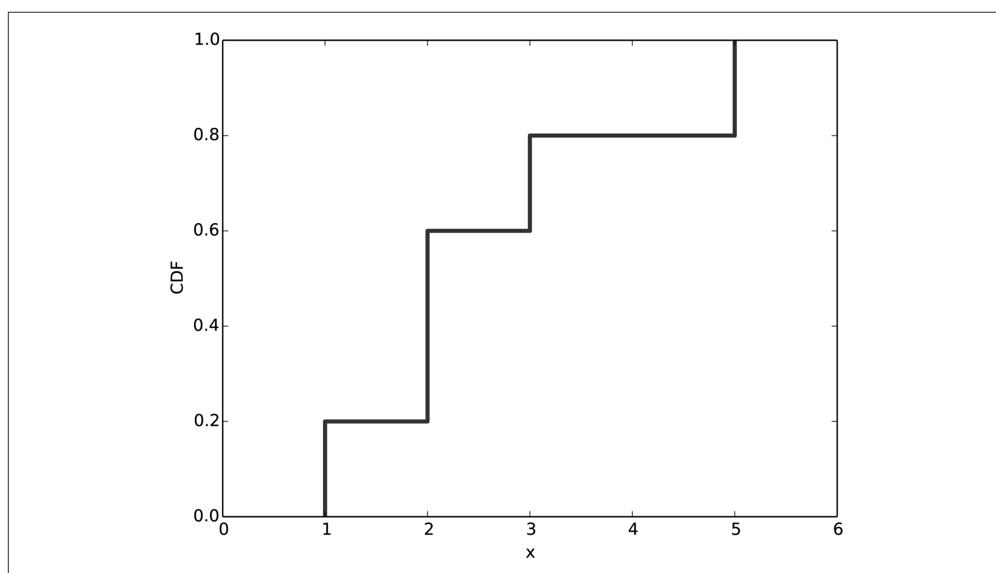



图 4-2：CDF 示例

# 4.4　表示CDF

`thinkstats2` 中有一个表示 CDF 的类 Cdf。Cdf 提供如下基本方法。

- `Prob(x)`

  对给定的值 `x`，计算其概率 $p$=CDF($x$)。方括号操作符等同于 `Prob` 方法。

- `Value(p)`

  对给定的概率 `p`，计算对应的值 `x`，即 `p` 的 CDF 反函数（inverse CDF）。

Cdf 的构造函数参数可以是一列值、一个 pandas Series、Hist、Pmf，或另一个 Cdf 对象。

下面的代码创建了全国家庭增长调查中妊娠期时间分布的 Cdf：

```
live, firsts, others = first.MakeFrames()
cdf = thinkstats2.Cdf(live.prglngth, label='prglngth')
```

thinkplot 提供一个函数 Cdf，可以将 Cdf 绘制为折线图。

```
thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='weeks', ylabel='CDF')
```

图 4-3 展示了绘制结果。解读 CDF 的方法之一是寻找其百分位数。例如，从图中我们可以看出，大约 10% 的妊娠期不超过 35 周，大约 90% 不超过 41 周。CDF 还展现了分布的形状。分布中经常出现的值在 CDF 中显示为陡峭或竖直的折线。在图 4-3 中，我们可以明显看出位于 39 周的众数。图中小于 30 周的值很少，因此 30 周左侧的折线很平缓。

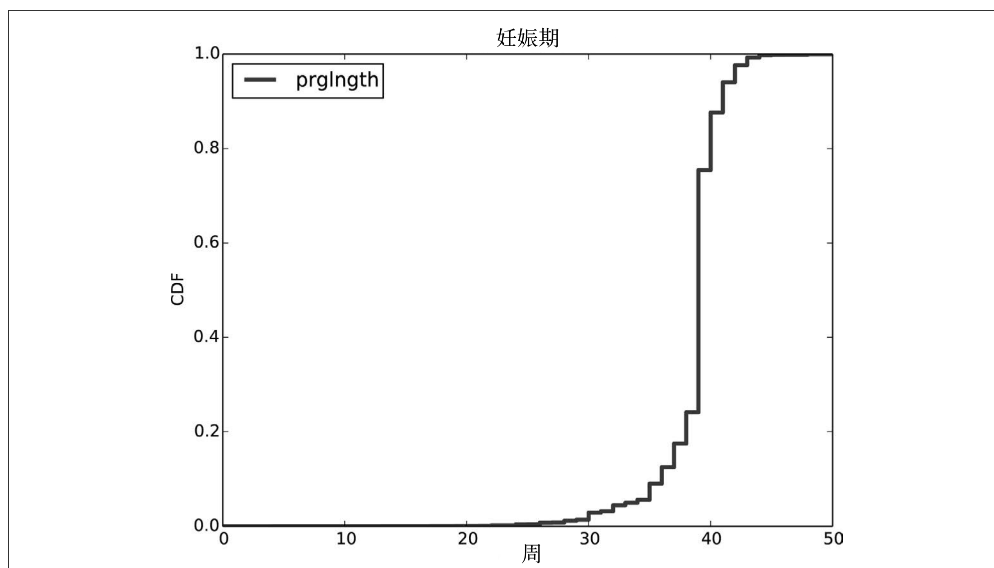



图 4-3：妊娠期时间 CDF

熟悉 CDF 可能需要一定的时间，但一旦了解，你就会发现 CDF 比 PMF 展示的信息更多，也更清晰。

## 4.5　比较CDF

在进行分布比较时，CDF 尤为有用。例如，下面一段代码绘制了第一胎和其他胎新生儿体重的 CDF。

```
first_cdf = thinkstats2.Cdf(firsts.totalwgt_lb, label='first')
other_cdf = thinkstats2.Cdf(others.totalwgt_lb, label='other')
```

```
thinkplot.PrePlot(2)
thinkplot.Cdfs([first_cdf, other_cdf])
thinkplot.Show(xlabel='weight (pounds)', ylabel='CDF')
```

图 4-4 展示了绘制结果。与图 4-1 相比较，图 4-4 更清晰地展示了分布的形状以及分布之间的差异。从图中我们可以看出，第一胎新生儿普遍体重较轻，而且大于均值时差异更为明显。

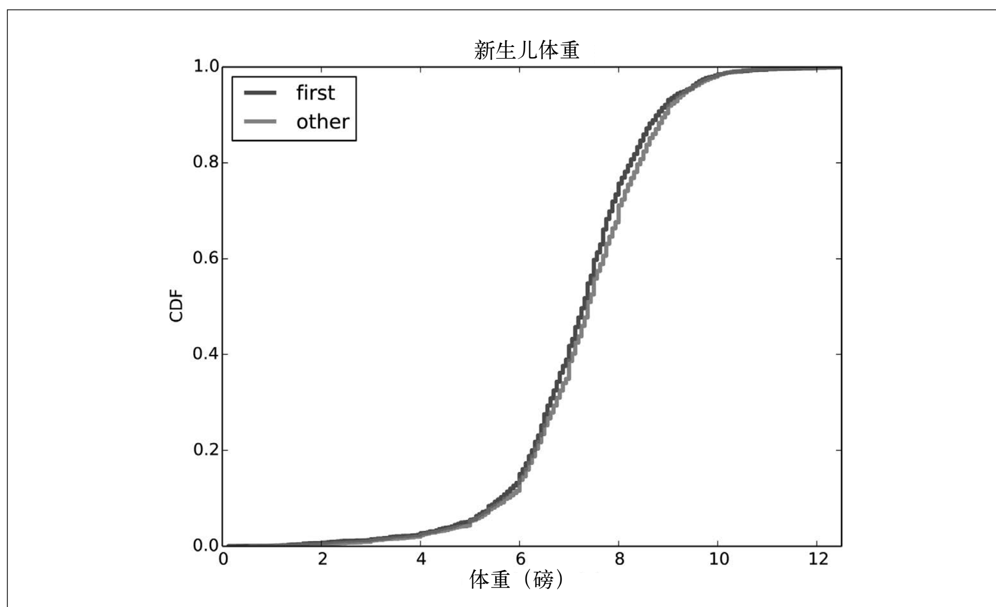



图 4-4：第一胎和其他情况下新生儿体重的 CDF

## 4.6　基于百分位数的统计量

计算出 CDF 之后，就很容易计算出百分位数和百分位秩。Cdf 类为此提供了两个方法。

- `PercentileRank(x)`

  对给定的值 `x`，计算其百分位秩，即 100 · CDF($x$)。

- `Percentile(p)`

  对给定的百分位秩 `rank`，计算对应的值 `x`。等价于 `Value(p/100)`。

`Percentile` 可以用于计算基于百分位数的汇总统计量。例如，第 50 百分位是将一个分布划分为两部分的值，也称为中位数（median）。和均值一样，中位数也是对分布集中趋势的度量。

实际上，“中位数”的定义有好几种，每种都具有不同的属性。但是 `Percentile(50)` 的定义非常简单，而且易于计算。

另一个基于百分位数的统计量是*四分位距*（interquartile range，IQR），用于度量一个分布的展布。四分位距是第 75 百分位和第 25 百分位的差值。

通常情况下，百分位数还用于对分布的形状进行简要描述。例如，人们经常用“五等份分组”描述收入分布，即按第 20、40、60 和 80 百分位数来分组。其他分布则划分为“十等份”。这种 CDF 中的等份点称为*分位数*（quantile）。要了解关于分位数的更多信息，请参考 https://en.wikipedia.org/wiki/Quantile。

## 4.7 随机数

假设我们要从成功生产的总体中选择一个随机样本，并查找样本中新生儿体重的百分位秩。如果我们计算出了这个百分位秩的 CDF，那这个分布会是什么样的呢？

为计算这个分布，首先要创建新生儿体重的 Cdf。

```
weights = live.totalwgt_lb
cdf = thinkstats2.Cdf(weights, label='totalwgt_lb')
```

然后，生成一个样本，并计算样本中每个值的百分位秩。

```
sample = np.random.choice(weights, 100, replace=True)
ranks = [cdf.PercentileRank(x) for x in sample]
```

`sample` 是新生儿体重的一个随机样本，样本大小为 100，使用*放回*（replacement）抽样，即同一个值可以选择多次。`ranks` 是百分位秩的一个列表。

最后，创建这些百分位秩的 Cdf 并进行绘制。

```
rank_cdf = thinkstats2.Cdf(ranks)
thinkplot.Cdf(rank_cdf)
thinkplot.Show(xlabel='percentile rank', ylabel='CDF')
```

图 4-5 展示了绘制结果。图中的 CDF 接近一条直线，说明这个分布是均匀的。

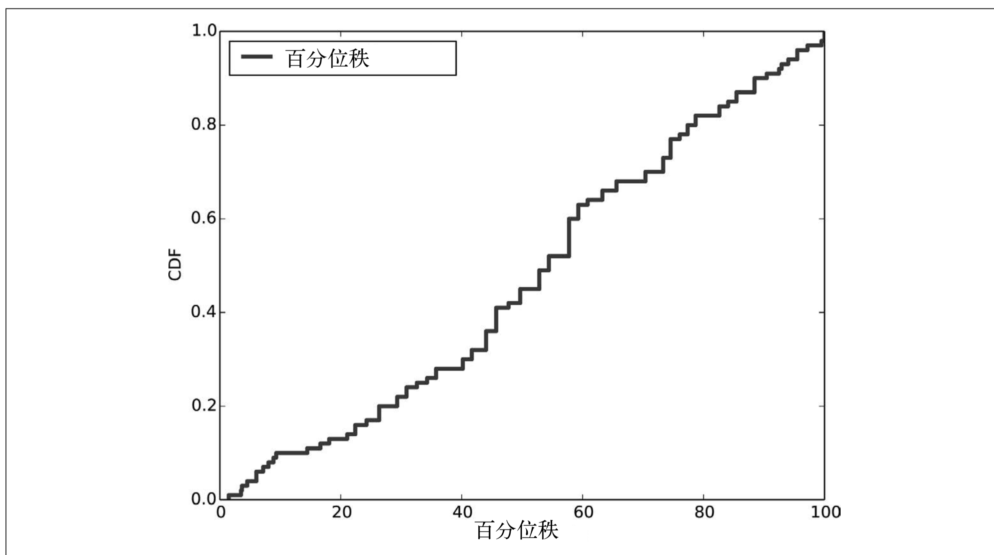



图 4-5：新生儿体重随机样本的百分位秩 CDF

这个结果可能不那么明显，但这是由 CDF 的定义方式决定的。这个图形显示，样本中 10% 的值小于第 10 百分位数，20% 的值小于第 20 百分位数，以此类推，完全符合预期。

因此，无论 CDF 的形状如何，其百分位秩的分布都是均匀的。这个属性非常有用，可以作为一种简单有效算法的基础，使用给定的 CDF 生成随机数。具体方法如下：

- 从 0 到 100 中均匀地选择一个百分位秩；
- 使用 `Cdf.Percentile`，得到分布中对应所选百分位秩的值。

Cdf 提供一个方法 `Random`，可实现上述算法。

```
# cass Cdf:
   def Random(self):
       return self.Percentile(random.uniform(0, 100))
```

Cdf 还提供一个方法 `Sample`，其参数为整数 `n`，返回从 Cdf 中随机选择的 `n` 个值列表。

## 4.8 比较百分位秩

百分位秩可以用来比较不同群组之间的度量值。例如，跑步比赛的选手通常按年龄和性别分组。要比较不同年龄组的选手，可以将比赛成绩转换成百分位秩。

几年前，我参加了在麻省 Dedham 举行的 James Joyce Ramble 10 公里比赛，成绩为 42:44，在 1633 名选手中排名 97。我超过或平了 1633 名选手中的 1537 名，因此在整个比赛中的百分位秩为 94%。

给定排名和参赛人数，我们可以计算出百分位秩。

```
def PositionToPercentile(position, field_size):
    beat = field_size - position + 1
    percentile = 100.0 * beat / field_size
    return percentile
```

我所在年龄组为 M4049，即“40~49 岁的男性”。在 M4049 中的 256 名选手中，我的排名为 26。因此，我在所属年龄组中的百分位秩为 90%。

如果 10 年之后我还在跑步（希望如此），将会分在 M5059 组中。假设我在所属年龄组中的百分位秩依然不变，那么会比现在的成绩慢多少？

要回答这个问题，我可以将自己在 M4049 中的百分位秩转换成 M5059 中的排名，从而得到答案。实现代码如下：

```
def PercentileToPosition(percentile, field_size):
    beat = percentile * field_size / 100.0
    position = field_size - beat + 1
    return position
```

M5059 组中有 171 名选手，因此我的成绩必须位于第 17 位或第 18 位成绩之间才能保持现在的百分位秩。M5059 组中第 17 名的比赛成绩是 46:05，所以我必须跑进 46:05 才能保持 90% 的百分位秩。

## 4.9 练习

你可以以 chap04ex.ipynb 为基础，进行下面的练习。本章练习的参考答案位于 chap04soln.ipynb 中。

- *练习 4.1*

你出生时的体重是多少？如果不知道，请打电话问问你的妈妈或者别的知情人。请使用全国家庭增长调查（所有成功生产）的数据，计算新生儿体重的分布，并用此分布算出你所在的百分位秩。如果你是第一个孩子，请在第一胎的新生儿体重分布中计算出你的百分位秩；否则请使用非第一胎的分布数据。如果你的百分位秩等于或大于 90，请打电话给你的妈妈，为自己给她带来的麻烦道歉。

- *练习 4.2*

`random.random` 生成的随机数应该均匀分布在 0 和 1 之间，即此范围中的每个值被选中的概率都应该相同。

请使用 `random.random` 生成 1000 个随机数，绘制这些数的 PMF 和 CDF。这些随机数的分布是均匀的吗？

## 4.10 术语

- 百分位秩（percentile rank）
  一个分布中小于或等于给定值的百分比。

- 百分位数（percentile）
  与给定百分位秩序相关联的值。

- 累积分布函数（cumulative distribution function，CDF）
  将值映射到累积概率的函数。CDF($x$) 是样本中小于或等于 $x$ 的值所占的比例。

- CDF 反函数（inverse CDF）
  从累积概率 $p$ 映射到对应值的函数。

- 中位数（median）
  位于第 50 百分位的值，经常用于度量集中趋势。

- 四分位距（interquartile range）
  第 75 百分位数和第 25 百分位数之间的差异，用于度量展布。

- 分位数（quantile）
  对应于等距百分位秩的一列值。例如，一个分布的四分位数为第 25 百分位数、第 50 百分位数和第 75 百分位数。

- 放回（replacement）
  采样过程的一种属性。“放回”是指同一个值可以选择多次；“不放回”是指一个值一旦被选中，就从总体中移除。

# 第5章

# 分布建模

目前为止，我们使用的分布都是基于有限样本的经验观察，因此称为经验分布（empirical distribution）。

在经验分布之外还有分析分布（analytic distribution）。分析分布的 CDF 是一个数学函数。分析分布可以用作经验分布的建模。此处所说的模型（model）是一种简化，以去除不必要的细节。本章将介绍常用的分析分布，以及如何使用这些分析分布对各种数据建模。

本章代码位于 analytic.py 中。前言介绍了如何下载和使用本书代码。

## 5.1 指数分布

指数分布（exponential distribution）相对简单，所以我们先从指数分布开始。指数分布的 CDF 为：

$$\mathrm{CDF}(x) = 1 - e^{-\lambda x}$$

参数 $\lambda$ 决定了分布的形状。图 5-1 展示了当 $\lambda = 0.5$、1 和 2 时 CDF 的形状。

在现实世界中，如果我们观察一系列事件，对事件发生的时间间隔，即到达间隔（interarrival time）进行测量，可能会得到指数分布。如果事件在任意时间发生的可能性相同，到达间隔的分布就会近似一个指数分布。

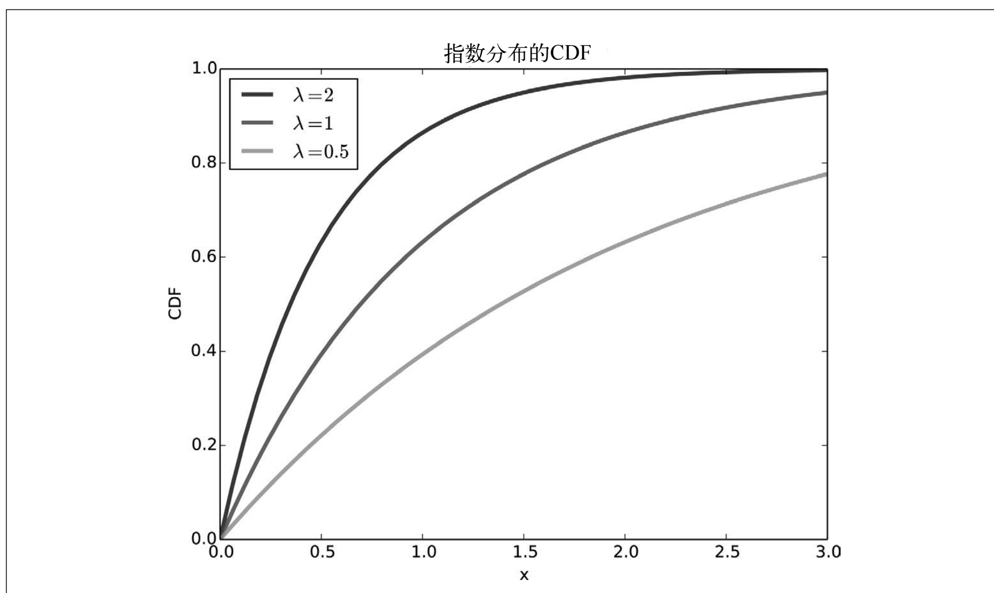



图 5-1：不同参数的指数分布的 CDF

举例说明，让我们来看看婴儿出生的到达间隔。1997 年 12 月 18 日，澳大利亚布里斯班的一家医院有 44 个婴儿出生[1]，当地报纸报道了这些婴儿的出生时间。ThinkStats2 代码库中的 babyboom.dat 文件包含了这 44 个婴儿出生时间的完整数据。

```
df = ReadBabyBoom()
diffs = df.minutes.diff()
cdf = thinkstats2.Cdf(diffs, label='actual')

thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='minutes', ylabel='CDF')
```

ReadBabyBoom 方法读取数据文件并返回一个 DataFrame 对象，对象的列名为 time、sex、weight_g 和 minutes，其中 minutes 是出生时间距离零点的分钟数。

diffs 是相邻出生时间的间隔，cdf 是这些到达间隔的分布。图 5-2（左）展示了 CDF。图中的分布看起来很像指数分布，但是我们如何确定呢？

一种方法是以 log-$y$ 为纵轴，绘制 CDF 补函数（complementary CDF，CCDF），即 1−CDF($x$)。对于指数分布的数据，绘制结果将是一条直线。让我们看看其中的原理。

注 1：这个示例的信息和数据来自 Dunn 的“A Simple Dataset for Demonstrating Common Distributions”，发表于 *Journal of Statistics Education* v.7, n.3 (1999)。

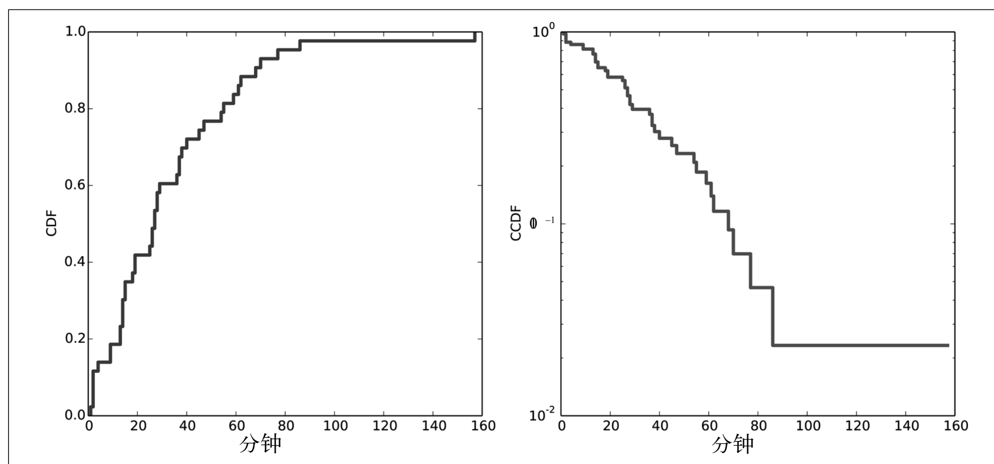



图 5-2：到达间隔 CDF（左）和以 log-$y$ 为纵轴的 CCDF（右）

如果你认为一个数据集是指数分布，那么绘制其 CDF 补函数时，预期看到的函数将是：

$$y \approx e^{-\lambda x}$$

两边取对数，得到：

$$\log y \approx -\lambda x$$

因此，如果以 log-$y$ 为纵轴，CCDF 函数将是一条斜率为 $-\lambda$ 的直线。我们可以使用如下代码生成这个图形：

```
thinkplot.Cdf(cdf, complement=True)
thinkplot.Show(xlabel='minutes',
               ylabel='CCDF',
               yscale='log')
```

指定参数 `complement=True` 后，`thinkplot.Cdf` 会在绘制图形之前计算 CDF 补函数。指定参数 `yscale='log'` 后，`thinkplot.Show` 会将 $y$ 轴设置为对数值。

图 5-2（右）展示了这段代码的结果。图中的线不是很直，说明指数分布并不是这组数据的完美模型。之前的假设——婴儿在一天中的任何时间出生的可能性相同，很可能并不成立。尽管如此，使用指数分布对这个数据集进行建模有可能是合理的。经过这种简化，我们用一个参数就可以概括这个分布。

参数 $\lambda$ 可以解释为一个比率，即事件在一个时间单元内发生的平均次数。在我们所举的例子中，24 小时内出生了 44 个婴儿，因此这个比率为 $\lambda$ 等于每分钟 0.030 6 个婴儿出生。指数分布的均值为 $1/\lambda$，故婴儿出生时间的时间间隔均值为 32.7 分钟。

## 5.2 正态分布

正态分布（normal distribution）也称为高斯分布，因其能近似描述很多现象而得到广泛使用。实际上，正态分布无处不在是有原因的，我们将在 14.4 节进行讨论。

正态分布由两个参数决定：均值 $\mu$ 和标准差 $\sigma$。$\mu = 0$ 且 $\sigma = 1$ 的正态分布称为标准正态分布（standard normal distribution）。标准正态分布的 CDF 是用积分定义的，没有封闭解，但有些算法可以进行有效的估算。SciPy 就提供了一种这样的算法：`scipy.stats.norm` 是一个表示正态分布的对象，提供方法 `cdf`，计算标准正态的 CDF。

```
>>> import scipy.stats
>>> scipy.stats.norm.cdf(0)
0.5
```

这个结果是正确的。标准正态分布的中位数是 0（与均值相同），分布中一半的值小于中位数，因此 CDF(0) 为 0.5。

`norm.cdf` 的可选参数 `loc` 可以指定均值，`scale` 可以指定标准差。

`thinkstats2` 提供一个更易于使用的函数 `EvalNormalCdf`，其参数为 `mu` 和 `sigma`，计算 `x` 的 CDF。

```
def EvalNormalCdf(x, mu=0, sigma=1):
    return scipy.stats.norm.cdf(x, loc=mu, scale=sigma)
```

图 5-3 展示了一组具有不同参数的正态分布的 CDF。这些曲线表现出的 S 形就是正态分布的显著特征。

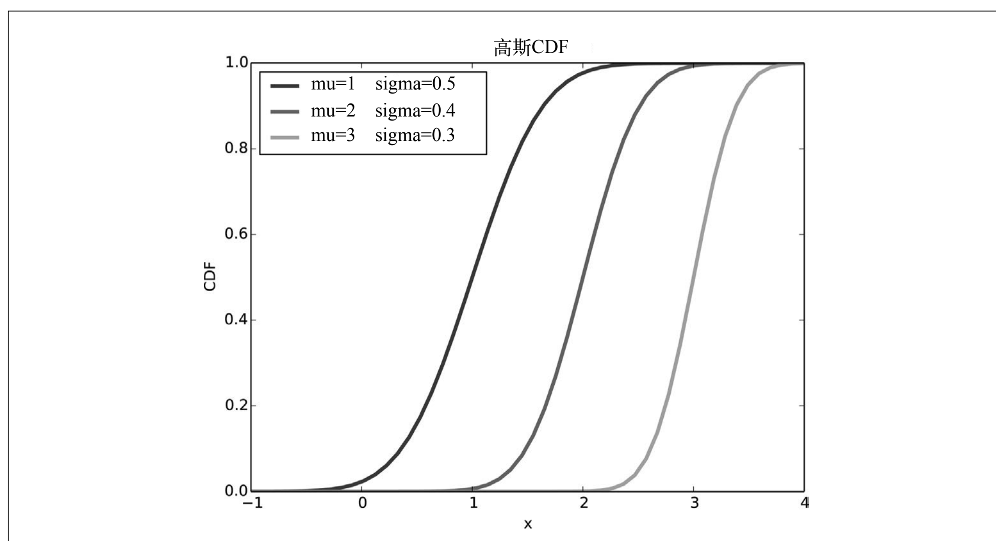



图 5-3：一组具有不同参数的正态分布的 CDF

在前一章，我们研究了全国家庭增长调查中新生儿体重的分布。图 5-4 展示了所有成功生产的新生儿体重的经验 CDF，以及具有相同均值和方差的正态分布的 CDF。

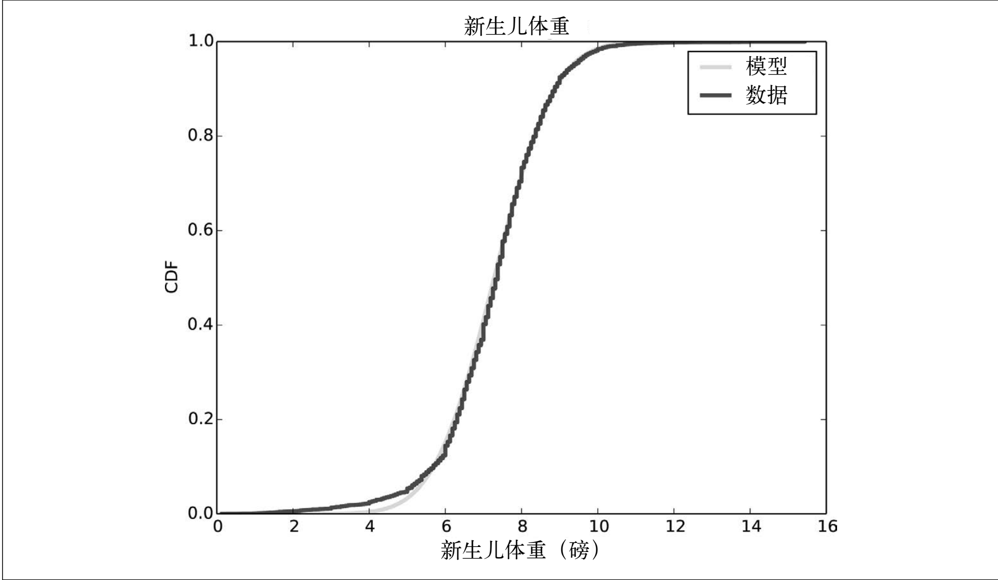



图 5-4：使用正态模型的新生儿体重 CDF

使用正态分布作为这个数据集的模型非常合适，因此如果我们用参数 $\mu = 7.28$ 和 $\sigma = 1.24$ 概括描述这个分布，结果误差（模型和数据之间的差）将会很小。

在低于第 10 百分位数的部分，数据和模型之间存在差距。相比正态分布中的期望值，实际数据中体重较轻的新生儿更多。如果我们要专门研究早产儿，就必须把这部分分布描述正确，因此使用正态模型未必合适。

# 5.3 正态概率图

对于指数分布和一些其他分析分布，我们可以通过简单转换来验证一个分析分布模型是否适用于一个数据集。

对于正态分布，则不存在这样的转换，但我们可以使用另外一种方法：正态概率图（normal probability plot）。有两种方法可生成正态概率图：困难方法和简单方法。如果你对困难方法感兴趣，请参考 https://en.wikipedia.org/wiki/Normal_probability_plot。简单方法如下：

(1) 将样本中的值排序；

(2) 从一个标准正态分布（$\mu$=0，$\sigma = 1$），生成一个随机样本并排序，样本大小与需要建模的样本一样；

(3) 绘制样本的排序值和随机值。

如果样本的分布接近正态分布，那么绘制结果将为一条直线，截距为 mu，斜率为 sigma。thinkstat2 提供一个方法 NormalProbability，参数为一个样本，返回两个 NumPy 数组：

```
xs, ys = thinkstats2.NormalProbability(sample)
```

ys 包含 sample 对象中排序后的值，xs 包含从标准正态分布得到的随机值。

为了测试 NormalProbability，要生成一些伪样本，这些样本实际上抽取自具有不同参数的正态分布。图 5-5 展示了结果。图中的线近似直线，尾部的值偏离程度较高。

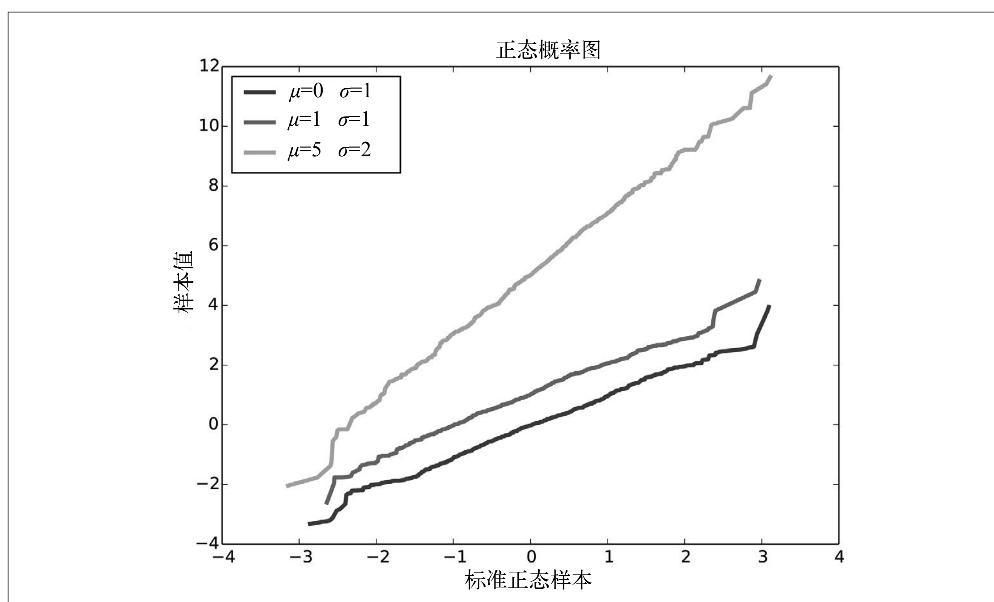



图 5-5：正态分布中随机样本的正态概率图

现在，让我们来试试使用真实数据。接下来的代码将为前一节中的新生儿数据生成正态概率图，其中灰色的线表示模型，蓝色的线表示真实数据。

```
def MakeNormalPlot(weights):
    mean = weights.mean()
    std = weights.std()

    xs = [-4, 4]
    fxs, fys = thinkstats2.FitLine(xs, inter=mean, slope=std)
    thinkplot.Plot(fxs, fys, color='gray', label='model')

    xs, ys = thinkstats2.NormalProbability(weights)
    thinkplot.Plot(xs, ys, label='birth weights')
```

`weights` 是一个 pandas Series 对象，代表新生儿体重。`mean` 和 `std` 分别表示均值和标准差。

`FitLine` 方法的参数为 `xs`、截距和斜率，返回值为 `xs` 和 `ys`，代表参数既定的直线在 `xs` 中所取的值。

`NormalProbability` 返回 `xs` 和 `ys`，包含了标准正态分布的值以及 `weights` 中的值。如果 `weights` 符合正态分布，那么数据应该与模型相符。

图 5-6 展示了所有成功生产及足月（孕期超过 36 周）生产的新生儿体重的正态概率图。两条曲线都在靠近均值的部分与模型相符，尾部出现偏差。最重的新生儿比模型预期的更重，最轻的新生儿则比模型预期的更轻。

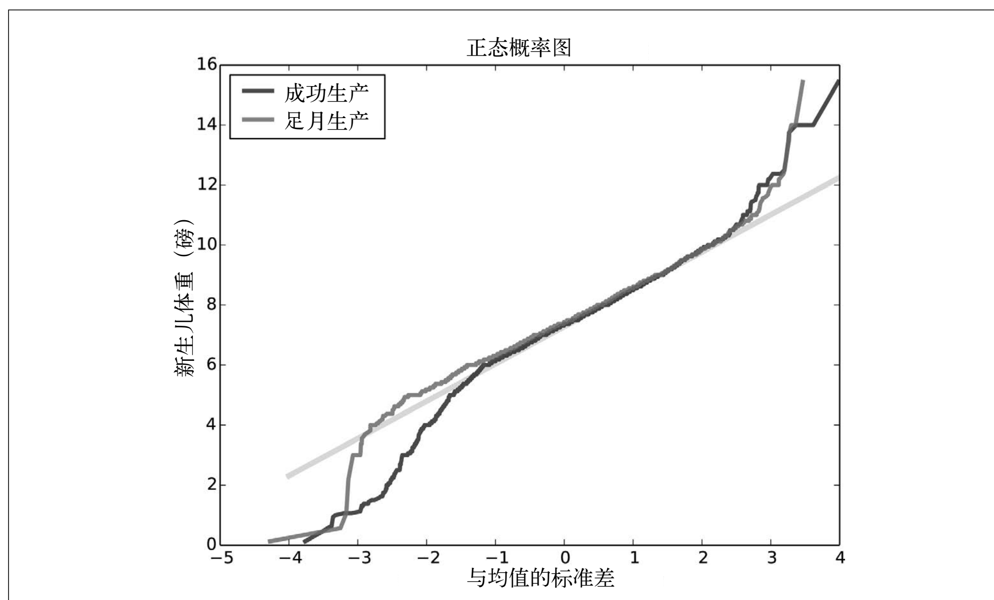



图 5-6：新生儿体重的正态概率图

如果我们只选择足月生产的数据，那么可以去除一部分最轻的体重，减少分布中左尾的偏差。

这张图说明，在距均值几个标准差的范围内，正态模型可以很好地描述样本数据的分布，但尾部偏差较大。至于这个模型能否满足实际需求，就要看具体需求究竟是什么了。

## 5.4 对数正态分布

如果一组值的对数符合正态分布，那么这组值就符合**对数正态分布**（lognormal distribution）。对数正态分布的 CDF 和正态分布的 CDF 一样，只不过将公式中的 $x$ 替换为了 $\log x$。

$$CDF_{lognormal}(x)=CDF_{normal}(\log x)$$

对数正态分布的参数通常写为 $\mu$ 和 $\sigma$。但是请记住，这两个参数并不是均值和标准差。对数正态分布的均值为 $\exp(\mu+\sigma^2/2)$，标准差的公式比较复杂，请参考 http://wikipedia.org/wiki/Log-normal_distribution。

如果一个样本近似对数正态分布，那么以 log−*x* 为纵轴绘制其 CDF，得到的图形会具备正态分布的特征。要测试样本与对数正态模型的拟合度，可以使用样本中值的 log 值绘制一个正态概率图。

举个例子，让我们来看看成年人的体重分布，这个分布大致符合对数正态分布。[1]

作为行为危险因素监测系统的一部分，国家慢性病预防和健康促进中心每年都会进行一项调查。[2] 2008 年，国家慢性病预防和健康促进中心调查了 414 509 名受访者，询问了他们的人口统计特征、健康状况和健康风险，在收集的数据中，就包括 398 484 名调查参与者的体重（单位为千克）。

本书代码库中包含 CDBRFS08.ASC.gz，这是一个等宽列的 ASCII 码文件，其中涵盖来自 BRFSS 的数据。代码库中还有一个 brfss.py 文件，能够读取数据文件并进行分析。

图 5-7（左）展示了使用线性刻度的成年人体重的分布和正态模型。图 5-7（右）使用对数刻度展示了同样的分布和对数正态模型。对数正态模型更符合成年人体重分布，但是我们在图 5-7 中可以看到两种模型在拟合度上的差别并不明显。

注 1：我最初是在 http://mathworld.wolfram.com/LogNormalDistribution.html 上看到成年人的体重符合对数正态分布，但是这个网页没有给出援引的出处。后来我找到了一篇论文，其中提出了数据转换的方法，并对原因进行了推测。这篇论文为 Penman and Johnson 的“The Changing Shape of the Body Mass Index Distribution Curve in the Population”，*Preventing Chronic Disease*, 2006 July; 3(3): A74。你可以在线阅读这篇论文：http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1636707。

注 2：Centers for Disease Control and Prevention (CDC). Behavioral Risk Factor Surveillance System Survey Data. Atlanta, Georgia: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, 2008.

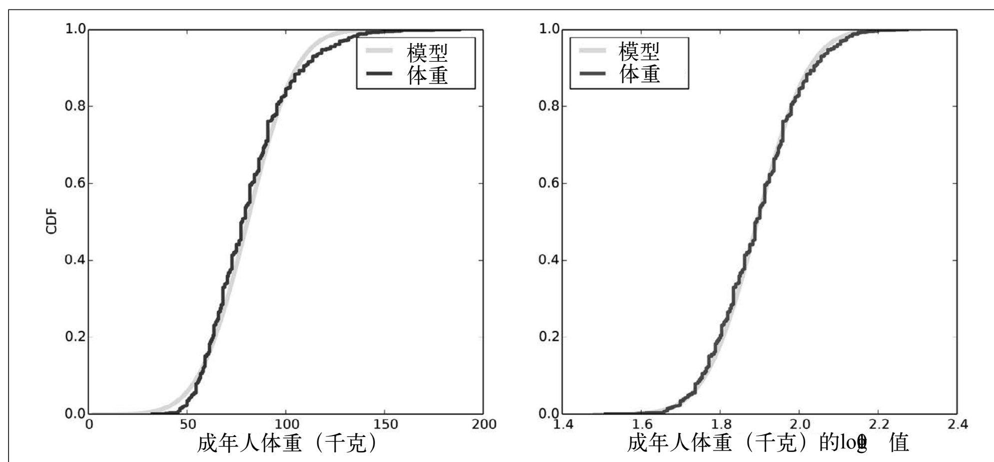



图 5-7：使用线性刻度（左）和对数刻度（右）的成年人体重 CDF

图 5-8 展示了成年人体重 $w$ 及其对数 $\log_{10}w$ 的正态概率图。在图 5-8 中，我们看到数据很明显地偏离了正态模型。在均值的几个标准差范围内，对数正态分布与数据吻合较好，但是在尾部也出现了偏离。我认为对数正态分布模型很好地模拟了成年人的体重数据。

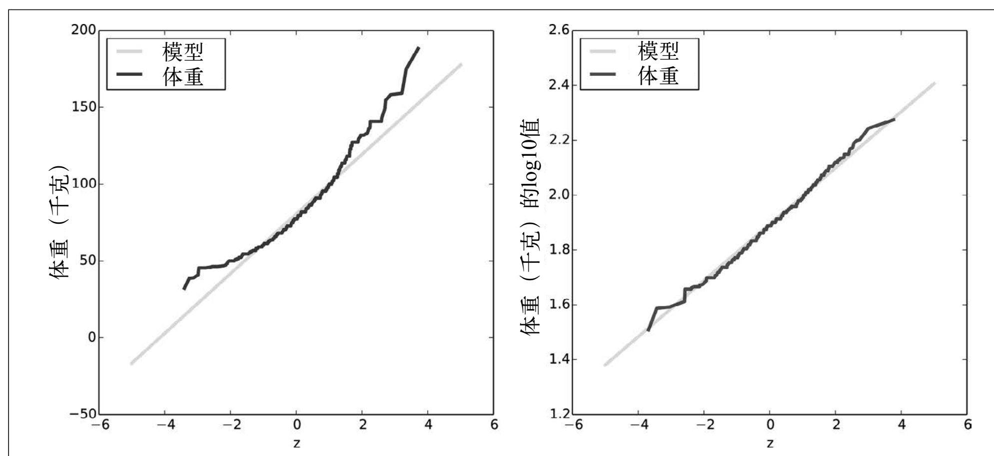



图 5-8：使用线性刻度（左）和对数刻度（右）的成年人体重正态概率图

## 5.5 Pareto分布

Pareto 分布（Pareto distribution）是以经济学家 Vilfredo Pareto 的名字命名的。Pareto 最初使用这个模型描述财富分布（参见 http://wikipedia.org/wiki/Pareto_distribution），之后人们用这个模型描述各种自然和社会科学现象，如城镇人口规模、沙粒和流星、森林火灾和地震。

Pareto 分布的 CDF 为：

$$\text{CDF}(x) = 1 - \left(\frac{x}{x_m}\right)^{-\alpha}$$

参数 $x_m$ 和 $\alpha$ 决定了分布的位置和形状。$x_m$ 是分布中可能出现的最小值。图 5-9 展示了 $x_m = 0.5$，具有不同 $\alpha$ 值的 Pareto 分布的 CDF。

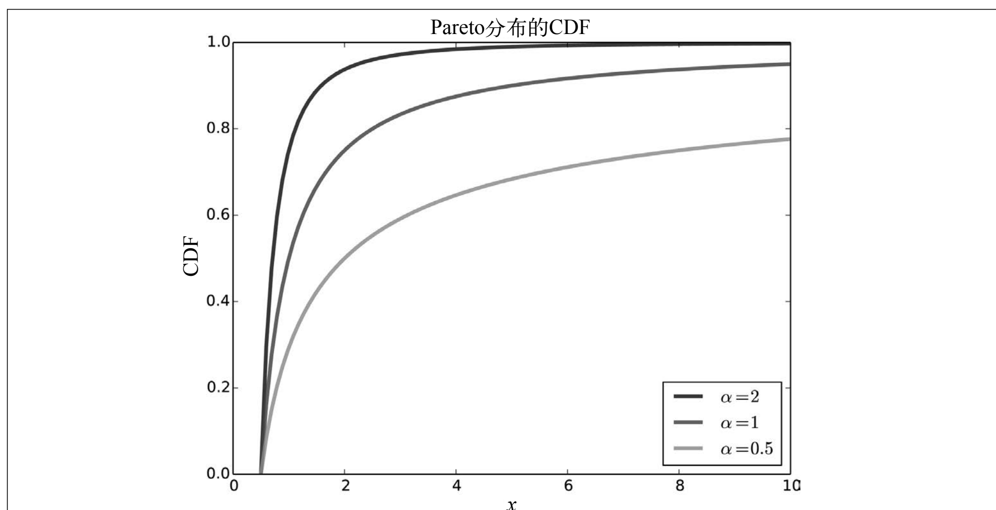



图 5-9：具有不同参数的 Pareto 分布的 CDF

有一种通过肉眼判断就能检验一个经验分布是否符合 Pareto 分布的简单方法，即如果横轴和纵轴都使用对数刻度，那么 CCDF 看起来会是一条直线。让我们看看其中的原理。

对于符合 Pareto 分布的一个样本，如果使用线性刻度绘制其 CCDF，那么预期会看到的函数如下：

$$y \approx \left(\frac{x}{x_m}\right)^{-\alpha}$$

公式两边取对数，可以得到：

$$\log y \approx -\alpha(\log x - \log x_m)$$

因此，如果以 $\log x$ 为横轴，$\log y$ 为纵轴，那么得到的函数图形会近似一条直线，斜率为 $-\alpha$，截距为 $\alpha \log x_m$。

让我们以城镇人口规模为例进行讨论。美国统计局（U.S. Census Bureau）负责发布美国每个城镇的人口数。

我 从 http://www.census.gov/popest/data/cities/totals/2012/SUB-EST2012-3.html 下 载 了 数据，放在本书代码库的 PEP_2012_PEPANNRES_with_ann.csv 文件中。代码库中还有一个 population.py 文件，负责读取数据文件，并绘制人口分布图。

图 5-10 展示了横轴和纵轴都使用对数刻度的的人口 CCDF。图中人口最多的 1% 城镇，即小于 $10^{-2}$ 的部分，基本是一条直线。因此，我们可以得出和其他一些研究者相同的结论，即城镇人口分布的尾部符合 Pareto 模型。

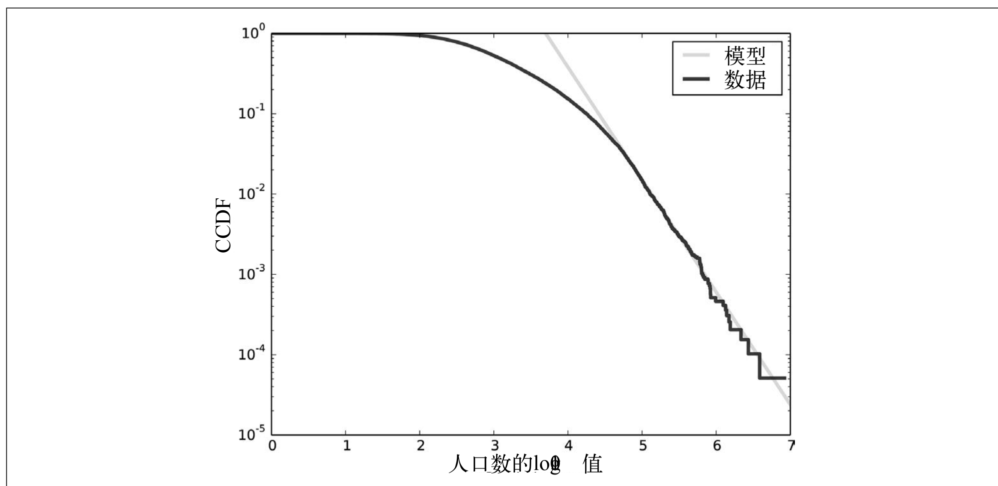



图 5-10：横轴和纵轴都使用对数刻度的城镇人口的 CCDF



另一方面，对数正态分布也可以很好地模拟城镇人口数据。图 5-11 展示了城镇人口规模的 CDF 和对数正态模型（左），以及正态概率图（右）。两个图中的数据和模型都相当吻合。

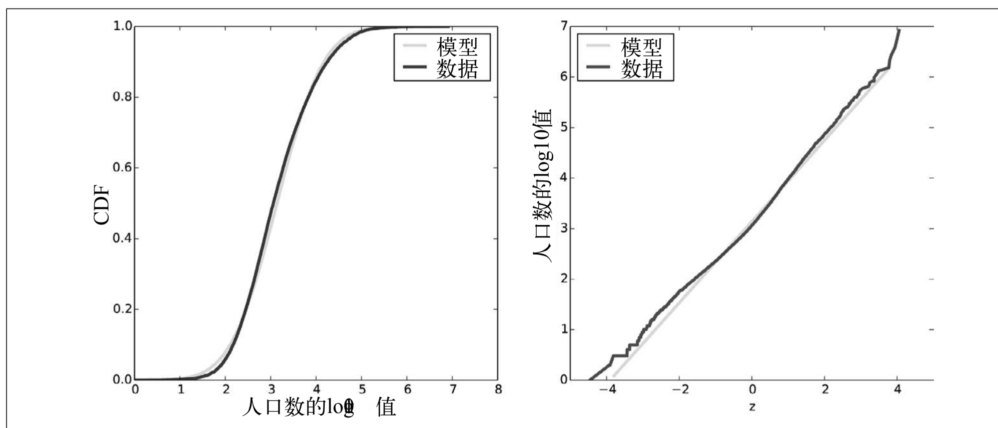



图 5-11：使用对数刻度横轴的城镇人口规模的 CDF（左），以及经过对数转换的人口规模的正态概率图（右）

这两个模型都不完美。Pareto 模型只适合人口规模最多的 1% 的城镇，但在这部分的拟合度更高。对数正态模型对其他 99% 的城镇拟合度更高。使用哪个模型更合适，取决于我们关注的是分布的哪一部分。

## 5.6 随机数生成

我们可以使用分析 CDF，根据给定的分布函数 $p = \mathrm{CDF}(x)$，生成随机数。如果我们有计算 CDF 反函数的有效方法，就可以从 0 到 1 的均匀分布中选择 $p$，然后得到 $x = \mathrm{ICDF}(p)$，从而为适当的分布生成随机数。

例如，指数分布的 CDF 为：

$$p = 1 - \mathrm{e}^{-\lambda x}$$

求解 $x$ 的公式为：

$$x = -\log(1 - p)/\lambda$$

因此，我们可以编写如下 Python 代码：

```
def expovariate(lam):
    p = random.random()
    x = -math.log(1-p) / lam
    return x
```

函数 `expovariate` 的参数为 `lam`，返回使用 `lam` 从指数分布中选取的一个随机数。

这段代码有两点需要注意。`lambda` 是 Python 关键字，因此我使用 `lam` 为参数名。另外，因为 log 0 没有定义，所以我们必须小心处理。`random.random` 返回值可以为 0，但不会为 1，因此 $1-p$ 可能为 1，但不会为 0，`log(1-p)` 不会出现没有定义的情况。

## 5.7 为什么使用模型

我在本章开头说过，现实世界的很多现象都可以用分析分布进行建模。你可能会问："那又如何？"

和所有模型一样，分析分布也是一种抽象，忽略了无关的细节。例如，一个观测分布可能带有测量误差或这个样本的独有特征，分析模型将这些特殊性都消除了。

分析模型也是一种数据压缩形式。如果模型很好地拟合了一个数据集，那么我们只需几个参数便可对大量的数据进行概括。

有时候，我们会惊讶地发现一种自然现象符合某个分析分布，而这些观察可以帮助人们更好地了解物理系统。有时我们可以对一个观测分布为何具有某种形态作出解释。例如，Pareto 分布经常是由带有正反馈的生成性过程（所谓的偏好依附过程，请参考 http://wikipedia.org/wiki/Preferential_attachment）导致的。

而且，分析分布很容易进行数学分析。第 14 章将会对此进行讨论。

但是，我们必须记住：所有模型都是不完美的。现实世界的数据永远不会完美地符合一个分析分布。人们谈论数据时，有时听众会感觉这些数据好像是由模型生成的。例如，人们可能会说人类的身高符合正态分布，或者收入符合对数正态分布。严格说来，这些说法不可能是正确的，现实世界和数学模型之间总是存在着差异。

如果模型描述了真实世界的相关特征，省略了不必要的细节，那么模型就是有用的。但是，什么是“相关”的，什么是“不必要”的，这取决于你将这个模型用于何种用途。

## 5.8　练习

你可以以 chap05ex.ipynb 为基础，进行下面的练习。本章练习的参考答案位于 chap05soln.ipynb 中。

- 练习 5.1

在 BRFSS（参见 5.4 节）中，身高分布大致符合正态分布。男性身高分布的 $\mu$ = 178 cm，$\sigma$ = 7.7 cm；女性身高分布的 $\mu$ = 163 cm，$\sigma$ = 7.3 cm。

如果你想加入 Blue Man Group，那么身高必须在 5 尺 10 寸 ~6 尺 1 寸（参见 http://bluemancasting.com）。美国男性人口中有百分之多少属于这个身高范围？提示：使用 `scipy.stats.norm.cdf`。

- 练习 5.2

为了体验一下 Pareto 分布，让我们来看看，如果人类身高符合 Pareto 分布，世界将会多么不同。使用参数 $x_m$ = 1 m，$\alpha$ = 1.7，我们得到一个 Pareto 分布，最小值为 1 m，中位数为 1.5 m。

请绘制这个分布。Pareto 世界中的人类身高均值是多少？身高未达到均值的人口比例是多少？如果 Pareto 世界中有 70 亿人，那么将有多少人高于 1 km ？最高的人将有多高？

- 练习 5.3

Weibull 分布是对在失败分析中出现的指数分布的一般化（参见 http://wikipedia.org/wiki/Weibull_distribution）。Weibull 分布的 CDF 为：

$$\mathrm{CDF}(x) = 1 - e^{-(x/\lambda)^k}$$

你能否找到一种转换方法，使 Weibull 分布的图形近似直线？这条线的斜率和截距各有什么含义？

请使用 `random.weibullvariate` 从一个 Weibull 分布中生成一个样本，测试你提出的转换方法。

- 练习 5.4

如果样本量 $n$ 很小，我们认为经验分布不能精确符合一个分析分布。要衡量样本与模型的拟合度，一个方法是从分析分布生成一个样本，检验所生成样本与数据的匹配程度。

例如，在 5.1 节中，我们绘制了出生时间到达间隔的分布，结果大致符合指数分布。但是这个分布的基础数据只有 44 个。为了判断这些数据是否来自指数分布，请以数据集的均值（即出生时间间隔为 33 分钟）为均值生成指数分布，从中得到 44 个值。

请绘制这些随机数的分布，并与实际分布进行比较。你可以使用 `random.expovariate` 生成这些值。

- 练习 5.5

本书代码库中有一组名为 mystery0.dat、mystery1.dat 等的数据文件。每个数据文件都包含由一个分析分布生成的一列随机数。

代码库中还有一个 test_models.py，这个脚本从文件读取数据，经过各种转换，绘制 CDF。你可以用如下命令运行这个脚本：

```
$ python test_models.py mystery0.dat
```

根据这些图形，你应该可以推断出哪种分布生成了哪个文件。如果觉得有困难，你可以查看 mystery.py，这个文件包含生成数据文件的代码。

- 练习 5.6

财富和收入分布有时使用对数正态分布和 Pareto 分布模型。要判断哪种模型更好，我们来看一些数据。

当前人口调查（Current Population Survey，CPS）是美国劳工统计局（Bureau of Labor Statistics）和美国人口调查局联合进行的收入和相关变量调查。2013 年的数据发布在 http://www.census.gov/hhes/www/cpstables/032013/hhinc/toc.htm 上。我下载了包含家庭收入信息的 hinc06.xls，并将这个 Excel 文件转换为 CSV 文件 hinc06.csv，这个文件位于本书的代码库中。代码库还包含读取这个 CSV 文件的 hinc.py。

请从这个数据集中得出收入分布。本章介绍的分析分布能很好地描述这些数据吗？参考答案位于 hinc_soln.py 中。

## 5.9 术语

- 经验分布（empirical distribution）
  一个样本中值的分布。

- 分析分布（analytic distribution）
  CDF 为分析函数的分布。

- 模型（model）
  一种实用的简化。分析分布通常是复杂经验分布的良好模型。

- 到达间隔（interarrival time）
  两个事件发生的时间间隔。

- CDF 补函数（complementary CDF）
  一个函数，将值 $x$ 映射到超过 $x$ 的值所占的比例，即 $1-\mathrm{CDF}(x)$。

- 标准正态分布（standard normal distribution）
  均值为 0 且均方差为 1 的正态分布。

- 正态概率图（normal probability plot）
  将样本值与标准正态分布中的随机值进行对比的图形。

# 第 6 章

# 概率密度函数

本章代码位于 density.py 中。前言介绍了如何下载和使用本书代码。

## 6.1 PDF

CDF 的导数称为概率密度函数（probability density function，PDF）。例如，指数分布的 PDF 公式如下：

$$\mathrm{PDF}_{\mathrm{expo}}(x) = \lambda e^{-\lambda x}$$

正态分布的 PDF 公式为：

$$\mathrm{PDF}_{\mathrm{normal}}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

对于特定值 $x$，人们通常不会计算其 PDF。计算 PDF 得到的不是概率，而是概率密度（density）。

在物理学上，密度是单位体积的质量。要计算质量，必须用密度乘以体积。如果密度不是常量，那么需要将其与体积进行积分。

类似地，概率密度度量单位 $x$ 的概率。为了计算概率，必须在 $x$ 的取值范围上进行积分。

`thinkstats2` 提供一个表示概率密度函数的 Pdf 类。每个 Pdf 对象都提供如下方法。

- `Density`

  参数为值 `x`，返回当前分布在 `x` 上的密度。

- `Render`

  对一组离散值计算密度，返回一个数对序列，数对中包含值 `xs` 及其概率密度 `ds`，序列按 `xs` 值排序。

- `MakePmf`

  对一组离散值计算 `Density`，返回标准化的 Pmf，结果近似 Pdf。

- `GetLinspace`

  返回 `Render` 和 `MakePmf` 默认使用的点集。

Pdf 是一个抽象的父类，不能实例化。也就是说，你不能创建一个 Pdf 对象，而应该定义一个继承 Pdf 的子类，并提供 `Density` 和 `GetLinspace` 的定义。Pdf 提供了 `Render` 和 `MakePmf` 的定义。

例如，`thinkstats2` 提供一个 `NormalPdf` 类，可用于计算正态密度函数。

```
class NormalPdf(Pdf):

    def __init__(self, mu=0, sigma=1, label=''):
        self.mu = mu
        self.sigma = sigma
        self.label = label

    def Density(self, xs):
        return scipy.stats.norm.pdf(xs, self.mu, self.sigma)

    def GetLinspace(self):
        low, high = self.mu-3*self.sigma, self.mu+3*self.sigma
        return np.linspace(low, high, 101)
```

NormalPdf 对象包含参数 `mu` 和 `sigma`。`Density` 方法使用 `scipy.stats.norm` 对象表示正态分布。`scipy.stats.norm` 提供很多功能，包括 `cdf` 和 `pdf` 方法（参见 5.2 节）。

下面一段代码以 BRFSS（参见 5.4 节）的成年女性身高（单位为厘米）的均值和方差为参数，创建一个 NormalPdf 对象，然后计算这个分布在距均值一个标准差处的密度。

```
>>> mean, var = 163, 52.8
>>> std = math.sqrt(var)
>>> pdf = thinkstats2.NormalPdf(mean, std)
>>> pdf.Density(mean + std)
0.0333001
```

代码执行结果为 0.03，单位为每厘米的概率值。需要再次声明，概率密度自身并没有太多含义。但是，如果绘制 Pdf，我们就能看出这个分布的形状。

```
>>> thinkplot.Pdf(pdf, label='normal')
>>> thinkplot.Show()
```

thinkplot.Pdf 绘制的 Pdf 是一个平滑函数，而 thinkplot.Pmf 绘制的 Pmf 是阶梯函数。图 6-1 展示了 thinkplot.Pdf 的绘制结果，以及从一个样本估计得到的 PDF，下一节将讨论如何计算。

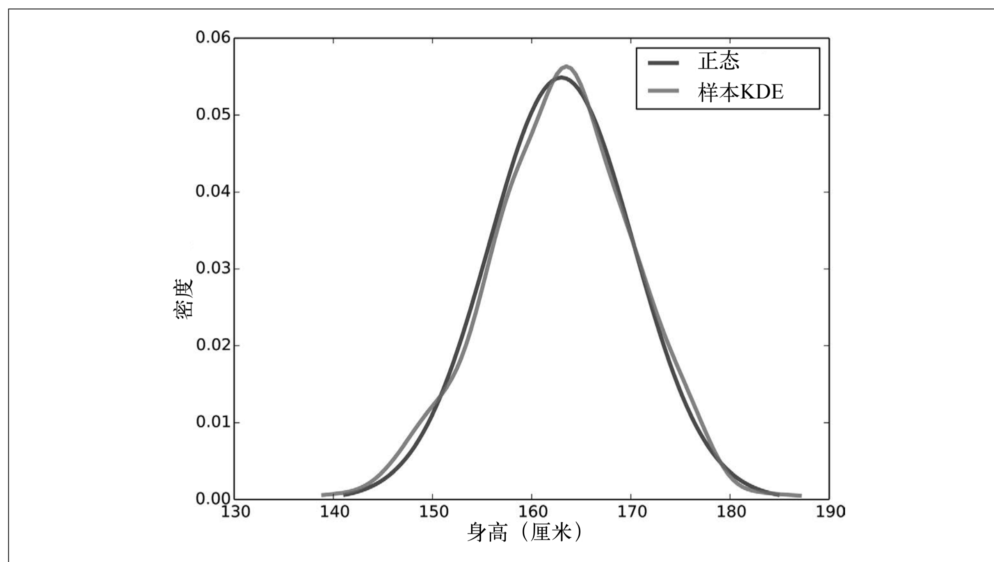



图 6-1：对美国成年女性身高进行建模得到的正态 PDF，以及样本量 $n$ 为 500 的核密度估计

你可以使用 MakePmf 模拟 Pdf。

```
>>> pmf = pdf.MakePmf()
```

默认情况下，MakePmf 得到的 Pmf 对象包含 101 个点，均匀分布在从 mu - 3*sigma 到 mu + 3*sigma 的范围内。MakePmf 和 Render 也可以使用关键字参数 low、high 以及点数 n。

## 6.2 核密度估计

核密度估计（kernel density estimation，KDE）是一种算法，可以对一个样本寻找符合样本数据的适当平滑的 PDF。你可以从 http://en.wikipedia.org/wiki/Kernel_density_estimation 了解到更多的信息。

scipy 提供了一种 KDE 实现，thinkstats2 中的 EstimatedPdf 类使用了 scipy 的 KDE 实现。

```
class EstimatedPdf(Pdf):

    def __init__(self, sample):
        self.kde = scipy.stats.gaussian_kde(sample)

    def Density(self, xs):
        return self.kde.evaluate(xs)
```

方法 `__init__` 以一个样本为参数，计算其核密度估计，得到一个 `gaussian_kde` 对象，这个对象提供 `evaluate` 方法。

方法 `Density` 可以以值或序列为参数，调用 `gaussian_kde.evaluate` 方法，返回得到的密度。类 `gaussian_kde` 使用了一个基于高斯分布的过滤器，使 KDE 变得平滑，因此类名中用了 Gaussian 这个词。

下面一段代码展示了如何从一个正态分布生成一个样本，然后创建一个符合该样本的 EstimatedPdf。

```
>>> sample = [random.gauss(mean, std) for i in range(500)]
>>> sample_pdf = thinkstats2.EstimatedPdf(sample)
>>> thinkplot.Pdf(pdf, label='sample KDE')
```

代码中的 `sample` 是一个列表，包含 500 个随机身高。`sample_pdf` 是一个 Pdf 对象，包含该样本的估计 KDE。`pmf` 是一个 Pmf 对象，通过计算在一个等距序列上的密度来模拟 Pdf。

图 6-1 展示了正态密度函数，以及基于 500 个随机身高样本的 KDE。估计结果与原始分布非常吻合。

KDE 估计密度函数可用于如下用途。

- 可视化
  在项目的探索阶段，展现分布的最佳方法通常是 CDF。在观察 CDF 之后，你可以判断估计 PDF 是否为该分布的适宜模型。如果 PDF 是适宜模型，那么就可以更好地向不熟悉 CDF 的受众展现这个分布。

- 插值
  通过估计 PDF，我们可以从样本得到总体模型。如果你相信总体分布是平滑的，那么就可以使用 KDE 为样本中不存在的值插入相应的密度。

- 模拟
  模拟通常是基于一个样本分布。如果样本规模较小，那么我们或许可以使用 KDE，对样本分布进行平滑处理，使得模拟可以探索可能性更高的结果，而不是复制所观察到的数据。

## 6.3　分布框架

到目前为止，我们已经学习了 PMF、CDF 和 PDF。让我们花点时间回顾一下。图 6-2 展示了这些函数之间的关系。

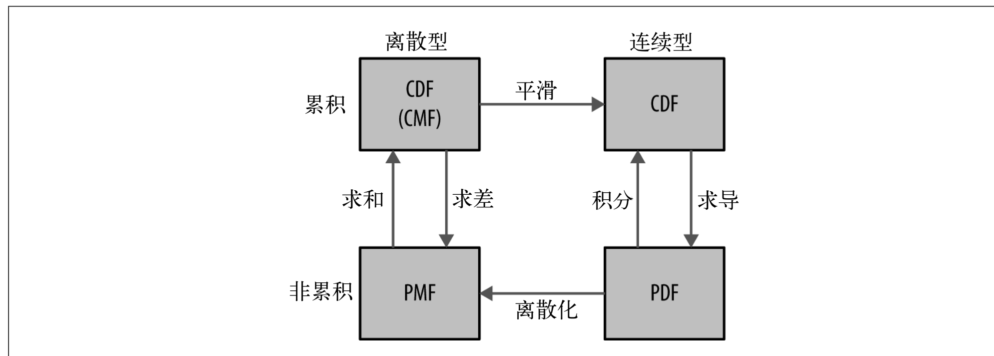



图 6-2：分布函数的关系框架

我们最先接触的是 PMF。PMF 代表一组离散值的概率。要从 PMF 得到 CDF，需要把概率值累加得到累积概率。反过来，要从 CDF 得到 PMF，需要计算累积概率之间的差值。我们将在接下来的几节讨论这些计算的具体实现。

PDF 是连续型 CDF 的导数，或者说，CDF 是 PDF 的积分。请记住：PDF 将值映射到概率密度，要得到概率，必须进行积分运算。

要从离散型分布得到连续型分布，可以进行各种平滑处理。一种平滑处理方法是，假设数据来自一个连续的分析分布（如指数分布或正态分布），然后估计这个分布的参数。另一种方法是核密度估计。

平滑处理的逆向操作是*离散化*（discretizing），或称为量化（quantizing）。如果在离散点上计算 PDF，就可以生成近似这个 PDF 的 PMF。使用数值积分可以获得更好的近似。

要明确区分连续型和离散型 CDF，也许将离散型 CDF 称为“累积质量函数”（cumulative mass function）更合适，但据我所知没有人使用这个词。

## 6.4　Hist实现

现在你应该已经知道如何使用 `thinkstats2` 中提供的各种基本类型（Hist、Pmf、Cdf 和 Pdf）了。接下来的几节将具体介绍这些基本类型的实现。这些内容也许可以帮助你更高效地使用这些类，但并不是必须了解的。

Hist 和 Pmf 继承了一个父类 `_DictWrapper`。这个类名开头的下划线表明这是一个“内部”

类，即这个类不应该在其他模块代码中使用。_DictWrapper 的名字说明了这个类的功能，它是一个字典封装类，其主要属性是将值映射到相关频次的字典 d。

字典 d 处理的值可以是任何能够进行散列计算的数据类型，值对应的频次应该为整数，但可以为任何数值类型。

_DictWrapper 提供适用于 Hist 和 Pmf 的方法，包括 __init__、Values、Items 和 Render。_DictWrapper 还提供修饰符方法 Set、Incr、Mult 和 Remove。这些方法都是使用字典运算实现的。示例如下：

```
# class_DictWrapper

    def Incr(self, x, term=1):
        self.d[x] = self.d.get(x, 0) + term

    def Mult(self, x, factor):
        self.d[x] = self.d.get(x, 0) * factor

    def Remove(self, x):
        del self.d[x]
```

Hist 还提供方法 Freq，用于查找一个给定值的频次。

Hist 的操作符和方法都是基于字典类的，因此这些方法都是常数时间操作，也就是说，这些方法的运行时间不会随着 Hist 对象变大而增加。

## 6.5 Pmf实现

Pmf 和 Hist 基本类似，唯一的区别是 Pmf 将值映射到浮点数的概率值，而不是整数的频次。如果一个 Pmf 的概率总和为 1，那么这个 Pmf 就是正态化的。

Pmf 提供一个方法 Normalize，这个方法计算所有概率的总和，并将所有概率除以一个因子。

```
# class Pmf

    def Normalize(self, fraction=1.0):
        total = self.Total()
        if total == 0.0:
            raise ValueError('Total probability is zero.')

        factor = float(fraction) / total
        for x in self.d:
            self.d[x] *= factor

        return total
```

参数 `fraction` 决定了正态化之后的概率总和，这个参数的默认值为 1。如果 Pmf 中的概率总和为 0，则无法进行正态化，因此这种情况下 `Normalize` 方法抛出一个 `ValueError`。

Hist 和 Pmf 的构造函数相同，参数都可以是 `dict` 对象、Hist 对象、Pmf 或 Cdf 对象、pandas 的 Series 对象、一列数值 - 频次对或一列数值。

如果实例化一个 Pmf，得到的结果将是正态化的。如果实例化一个 Hist，得到的结果不是正态化的。要构建一个非正态化的 Pmf，可以先创建一个空的 Pmf，然后进行修改。Pmf 修饰符不会对 Pmf 重新进行正态化。

## 6.6 Cdf实现

CDF 将值映射到累积概率，因此我原本可以用 `_DictWrapper` 实现 Cdf。但是 CDF 中的值是按序排列的，而 `_DictWrapper` 中的值是乱序的。而且，人们经常需要计算 CDF 反函数，也就是说，从累积概率映射到值。因此，我选择使用两个排序列表来实现 Cdf，以便使用二分法检索，在对数时间内进行正向或反向查找。

Cdf 构造函数的参数可以是一个值序列、pandas 的 Series 对象、将值映射到概率的字典、一列数值 - 频次对、Hist 对象、Pmf 对象或 Cdf 对象。如果向 Cdf 构造函数传入两个参数，那么构造函数会将这两个参数当作一个已排序的值序列及其对应的累积函数序列。

给定一个序列、pandas Series 对象或字典，Cdf 构造函数会创建一个 Hist 对象，然后使用这个 Hist 对象对属性进行初始化。

```
        self.xs, freqs = zip(*sorted(dw.Items()))
        self.ps = np.cumsum(freqs, dtype=np.float)
        self.ps /= self.ps[-1]
```

`xs` 是按序排列的值列表，`freqs` 是对应的频次列表。`np.cumsum` 计算这些频次的累积总和。将所有的累积频次除以频次总和就得到了累积概率。对于 `n` 个值，构造一个 Cdf 所需的时间与 $n\log n$ 成正比。

对于一个参数值，`Prob` 方法返回其累积概率。具体实现如下：

```
# class Cdf
    def Prob(self, x):
        if x < self.xs[0]:
            return 0.0
        index = bisect.bisect(self.xs, x)
        p = self.ps[index - 1]
        return p
```

`bisect` 模块实现了一种二分法检索。对于传入的累积概率参数，Cdf 的 `Value` 方法返回其对应的值，具体实现如下：

```
# class Cdf
    def Value(self, p):
        if p < 0 or p > 1:
            raise ValueError('p must be in range [0, 1]')

        index = bisect.bisect_left(self.ps, p)
        return self.xs[index]
```

给定一个 Cdf，我们可以计算相邻累积概率的差值，从而得到 Pmf。如果调用 Cdf 构造函数并传入一个 Pmf，构造函数会调用 `Cdf.Items` 方法计算相邻累积概率的差值。

```
# class Cdf
    def Items(self):
        a = self.ps
        b = np.roll(a, 1)
        b[0] = 0
        return zip(self.xs, a-b)
```

`np.roll` 将数组 `a` 中的元素向右移动一个位置，最后一个元素"滚动"（roll）到数组的起始位置。我们将 `b` 的第一个元素替换为 0，然后计算差值 `a-b`，得到的结果是包含概率数值的一个 NumPy 数组。

Cdf 提供方法 `Shift` 和 `Scale`，对 Cdf 中的值进行修改，但概率应视为不变。

## 6.7 矩

不管在什么时候，只要将一个样本归结为一个数字，这个数字就是一个统计量。到目前为止，我们接触到的统计量包括均值、方差、中位数和四分位距等。

*原始矩*（raw moment）也是一个统计量。对于一组值为 $x_i$ 的样本，第 $k$ 个原始矩计算公式为：

$$m'_k = \frac{1}{n}\sum_i x_i^k$$

Python 实现如下：

```
def RawMoment(xs, k):
    return sum(x**k for x in xs) / len(xs)
```

当 $k$=1 时，原始矩为样本的均值 $\bar{x}$。其余的原始矩本身并不具有任何意义，但可以用在一些计算中。

*中心矩*（central moment）的用途较多。第 $k$ 个中心矩的计算公式为：

$$m_k = \frac{1}{n}\sum_i (x_i - \bar{x})^k$$

Python 实现如下：

```
def CentralMoment(xs, k):
    mean = RawMoment(xs, 1)
    return sum((x - mean)**k for x in xs) / len(xs)
```

当 $k$=2 时，计算结果是第二中心矩，你可能会发现这其实就是方差。这些统计量为什么称为矩，方差的定义给了我们一些提示。如果我们在直尺的不同位置 $x_i$ 附加一个重物，然后将直尺围绕这些值的均值旋转，旋转重物的惯性力矩（moment of inertia）就是这些值的方差。如果不熟悉惯性力矩的概念，可以参考 http://en.wikipedia.org/wiki/Moment_of_inertia。

在使用基于矩的统计量时，很重要的一点是考虑统计量的单位。例如，如果值 $x_i$ 的单位是厘米，那么第一原始矩的单位也是厘米，但第二原始矩的单位是平方厘米，第三原始矩的单位是立方厘米，以此类推。

由于这些不同寻常的单位，矩本身的含义很难解释。正因为如此，人们通常会使用标准差而不是第二中心矩。标准差是方差的平方根，使用的单位与 $x_i$ 相同。

## 6.8 偏度

偏度（skewness）是描述分布形状的一个属性。如果分布是以集中趋势为中心对称的，那么这个分布就非偏斜的（unskewed）。如果分布中的值向右延伸更多，那么这个分布就是“右偏”（right skewed）的；如果值向左延伸更多，那么这个分布就是“左偏”（left skewed）的。

“偏斜”（skewed）并不是通常所说“有偏”（biased）的含义。偏度只是描述了分布的形状，和采样过程是否有偏并无关系。

人们通常使用几个统计量对分布的偏度进行量化。对给定的值序列 $x_i$，样本偏度（sample skewness）$g_1$ 的计算方法如下：

```
def StandardizedMoment(xs, k):
    var = CentralMoment(xs, 2)
    std = math.sqrt(var)
    return CentralMoment(xs, k) / std**k

def Skewness(xs):
    return StandardizedMoment(xs, 3)
```

$g_1$ 是第三标准化矩（standardized moment），也就是说这个值经过了正态化，没有单位。

偏度为负值代表一个分布左偏，偏度为正值代表一个分布右偏。$g_1$ 的大小代表偏斜的程度，但是 $g_1$ 的值本身很难解读。

在实际应用中，计算样本偏度通常并非好主意。分布中的任何离群值都会对 $g_1$ 产生不同程度的影响。

衡量分布对称性的另一个方法是检查均值和中位数的关系。极端值对均值的影响比对中位数影响更大，因此在一个左偏分布中，均值会比中位数小；在右偏分布中，均值则比中位数大。

Pearson 中位数偏度系数（Pearson's median skewness coefficient）是基于样本均值和中位数差的一种偏度度量。

$$g_p = 3(\bar{x} - m)/S$$

其中 $\bar{x}$ 是样本均值，$m$ 是中位数，$S$ 是标准差。Python 实现如下：

```
def Median(xs):
    cdf = thinkstats2.MakeCdfFromList(xs)
    return cdf.Value(0.5)

def PearsonMedianSkewness(xs):
    median = Median(xs)
    mean = RawMoment(xs, 1)
    var = CentralMoment(xs, 2)
    std = math.sqrt(var)
    gp = 3 * (mean - median) / std
    return gp
```

这个统计量是稳健的（robust），即受离群值的影响较小。

举个例子，让我们看看全国家庭增长调查妊娠数据中的新生儿体重的偏度。估计和绘制 PDF 的代码如下：

```
live, firsts, others = first.MakeFrames()
data = live.totalwgt_lb.dropna()
pdf = thinkstats2.EstimatedPdf(data)
thinkplot.Pdf(pdf, label='birth weight')
```

图 6-3 展示了代码的运行结果。图中左尾看起来比右尾长，因此我们猜测这个分布是左偏的。其均值 7.27 磅比中位数 7.38 磅略小，与左偏分布的特征一致。样本偏度为 −0.59，Pearson 中位数偏度系数为 −0.23，都是负值。

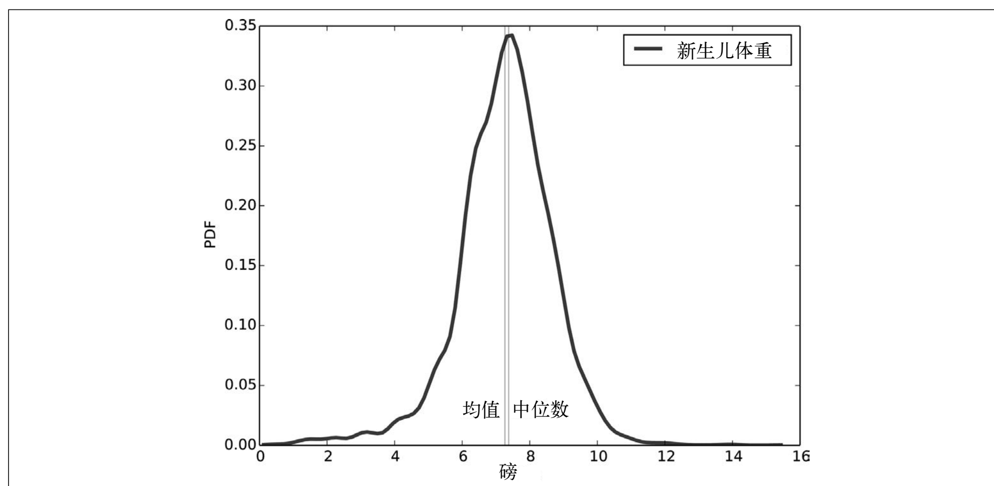



图 6-3：全国家庭增长调查新生儿体重数据的估计 PDF

现在我们将这个分布与 BRFSS 成年人体重分布进行比较。估计和绘制 BRFSS 成年人体重 PDF 的代码如下：

```
df = brfss.ReadBrfss(nrows=None)
data = df.wtkg2.dropna()
pdf = thinkstats2.EstimatedPdf(data)
thinkplot.Pdf(pdf, label='adult weight')
```

图 6-4 展示了这段代码的运行结果。图中的分布看起来向右偏斜。其均值 79.0 确实比中位数 77.3 大。样本偏度为 1.1，Pearson 中位数偏度系数为 0.26。

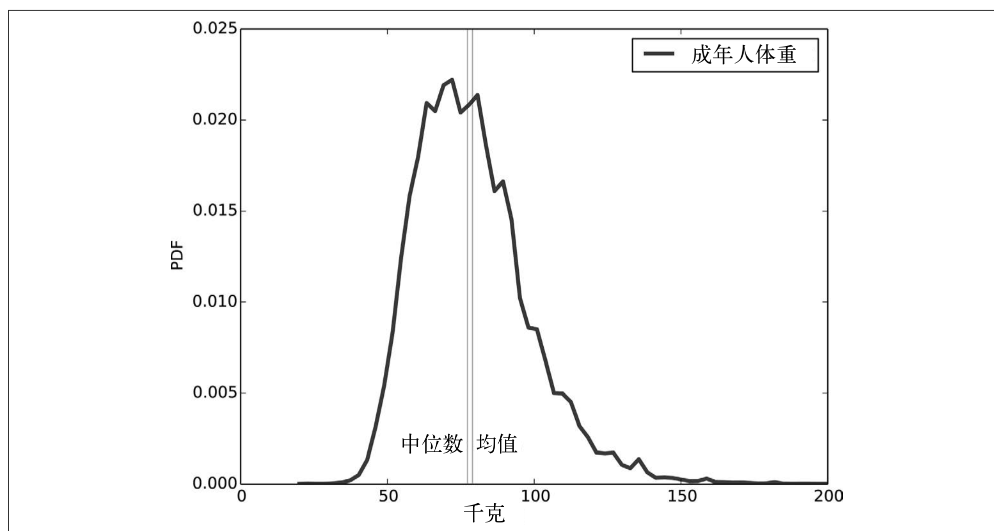



图 6-4：BRFSS 成年人体重数据的估计 PDF

偏度系数为正或为负说明了分布左偏还是右偏，但除此之外，我们很难对偏度系数作出更多解释。样本偏度的稳健性较差，即更容易受到离群值的影响，因此在应用于偏斜分布时可靠性更低，而这恰恰是人们最关心样本偏度的时候。

Pearson 中位数偏度系数基于计算所得的均值和方差，因此也会受到离群值影响，但这个系数不依赖第三矩，因此稳健性稍好一些。

## 6.9 练习

本章练习的参考答案位于 chap06soln.py 中。

- 练习 6.1

众所周知，收入分布向右偏斜。在这个练习中，我们将度量收入分布的偏度有多大。

当前人口调查是美国劳工统计局和美国人口调查局联合进行的一项调查，用于研究收入及其相关变量。这项调查 2013 年收集的数据位于 http://www.census.gov/hhes/www/cpstables/032013/hhinc/toc.htm。我下载了包含家庭收入信息的 hinc06.xls 文件，并将其转换为 CSV 文件 hinc06.csv，放于本书的代码库中。代码库还包含一个 hinc2.py，用于读取 hinc06.csv，并对数据进行转换。

这个数据集包含一系列的收入范围，以及位于各范围中的调查参与者的人数。最低收入范围中的调查参与者家庭年收入“低于 5000 美元”。最高收入范围中的参与者家庭年收入为“等于或超过 250 000 美元”。

为了估计这些数据的均值以及其他统计量，我们必须对收入上限、下限，以及数据在各个收入范围内的分布进行一些假设。hinc2.py 中的 `InterpolateSample` 方法提供了对这些数据建模的一种方法。`InterpolateSample` 的一个参数是 DataFrame 对象，这个 DataFrame 对象中的 `income` 列包含每个收入范围的上限，`freq` 列包含每个收入范围内的调查参与者人数。

`InterpolateSample` 的另一个参数 `log_upper` 是对最高收入范围的一个设定上限，表示为收入值的 `log10` 美元。默认值 `log_upper=6.0` 代表对调查参与者的最高收入设定的上限为 $10^6$，即 100 万美元。

`InterpolateSample` 生成一个伪样本，也就是说，产生的这个家庭收入样本中，各个收入范围的调查参与者数量与实际数据相同。`InterpolateSample` 假定每个范围内的收入的 log10 是均匀分布的。

请计算 `InterpolateSample` 所生成样本的中位数、均值、偏度和 Pearson 偏度系数。纳税收入低于均值的家庭所占比例为多少？这个结果与设定的收入上限有怎样的依赖关系？

## 6.10 术语

- 概率密度函数（probability density function，PDF）
  连续 CDF 的导数。这个函数将值映射到其概率密度。

- 概率密度（probability density）
  一个数值，可以在一个取值范围上进行积分得到一个概率。如果值的单位为厘米，那么概率密度的单位为每厘米的概率。

- 核密度估计（kernel density estimation，KDE）
  基于一个样本对 PDF 进行估计的算法。

- 离散化（discretize）
  用离散函数对一个连续函数或分布进行模拟。离散化的逆操作是平滑化。

- 原始矩（raw moment）
  一个统计量，基于数据乘方之和。

- 中心矩（central moment）
  一个统计量，基于数据与均值差的乘方之和。

- 标准化矩（standardized moment）
  矩的一个比率，没有单位。

- 偏度（skewness）
  度量分布的对称性。

- 样本偏度（sample skewness）
  一个基于矩的统计量，用于量化分布的偏度。

- Pearson 中位数偏度系数（Pearson's median skewness coefficient）
  用于量化分布偏度的一个统计量，基于中位数、均值和标准差。

- 稳健（robust）
  如果一个统计量受离群值的影响相对较小，那么这个统计量就是稳健的。

# 第 7 章
# 变量之间的关系

到目前为止，我们每次只研究一个变量。本章，我们将研究变量之间的关系。如果能够从一个变量的信息中得到另一个变量的信息，那么这两个变量就是相关的。例如，身高和体重是相关的，个子高的人一般体重也会比较重。当然，身高和体重的关系并不是绝对的，也有矮个的胖子和高个的瘦子。但是在猜测一个人的体重时，如果已知身高的话，进行推测会更加准确。

本章代码位于 scatter.py 中。前言介绍了如何下载和使用本书代码。

## 7.1 散点图

研究两个变量之间关系的最简单方法是*散点图*（scatter plot）。但好的散点图的绘制并不简单。作为示例，我将绘制 BRFSS（参见 5.4 节）调查参与者的体重与身高关系的散点图。

如下代码可读取数据文件，获得身高和体重数据。

```
df = brfss.ReadBrfss(nrows=None)
sample = thinkstats2.SampleRows(df, 5000)
heights, weights = sample.htm3, sample.wtkg2
```

`SampleRows` 选出这些数据的一个随机子集。

```
def SampleRows(df, nrows, replace=False):
    indices = np.random.choice(df.index, nrows, replace=replace)
    sample = df.loc[indices]
    return sample
```

df 是 DataFrame 对象；nrows 是需要选取的行数；replace 是一个布尔值，指定采样过程是否放回，也就是说，同一列是否可以被多次选中。

thinkplot 提供 Scatter 方法绘制散点图。

```
thinkplot.Scatter(heights, weights)
thinkplot.Show(xlabel='Height (cm)',
               ylabel='Weight (kg)',
               axis=[140, 210, 20, 200])
```

绘制结果展示了身高和体重的关系形状，见图 7-1（左）。正如我们预期的那样，个子高的人体重更重。

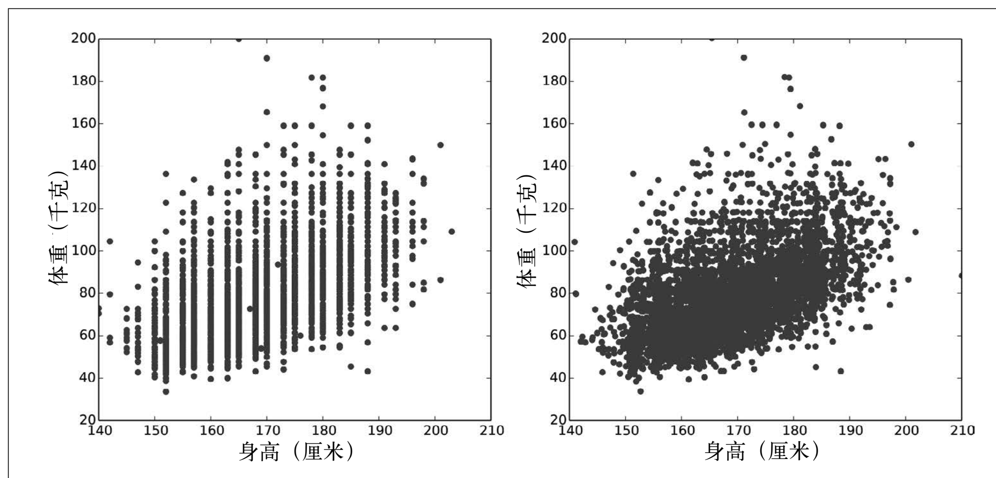



图 7-1：BRFSS 调查参与者的体重与身高关系散点图，未抖动（左），抖动（右）

但这个绘制结果的展现效果并不理想，因为数据都成列聚集。之所以产生这个现象，是因为身高数据四舍五入到相邻的英寸，转换为厘米后，再次四舍五入。在这个转换过程中，丢失了一些信息。

我们无法找回已丢失的信息，但可以将数据进行*抖动*（jittering），即加入随机噪音弥补四舍五入的效果，以减少丢失信息对散点图的影响。这些测量数据都四舍五入到相邻的英寸，因此导致的偏差可能达到 0.5 英寸（或 1.3 厘米）。同样，体重数据的偏差最大可能达到 0.5 千克。

```
heights = thinkstats2.Jitter(heights, 1.3)
weights = thinkstats2.Jitter(weights, 0.5)
```

Jitter 方法的具体实现如下：

```
def Jitter(values, jitter=0.5):
```

```
    n = len(values)
    return np.random.uniform(-jitter, +jitter, n) + values
```

参数 values 可以是任何序列，结果为 NumPy 数组。

图 7-1（右）展示了抖动处理之后的绘制结果。抖动减弱了四舍五入导致的视觉效果，使变量关系的形状更加清晰。但是，抖动数据通常只应用于视觉效果，你应该避免在分析时使用经过抖动处理的数据。

即便经过了抖动处理，散点图也不是展示数据的最佳方法。图中有很多重叠的点，遮盖了密集部分的数据，使离群值显得特别突出。这种效果称为饱和（saturation）。

我们可以使用参数 alpha 解决这个问题，将图中的点显示为半透明的。

```
thinkplot.Scatter(heights, weights, alpha=0.2)
```

图 7-2（左）展示了绘制结果。重叠的数据点颜色较深，因此颜色的深度与数据的密集程度成正比。在这张图中，我们可以看到两个新的细节：竖直方向上，数据点汇集在几个高度周围；接近 90 千克（或 200 磅）处有一条水平线。这些数据是调查参与者自己提供的，单位为磅，因此很有可能是因为一些参与者提供了四舍五入的数据从而导致这些现象。

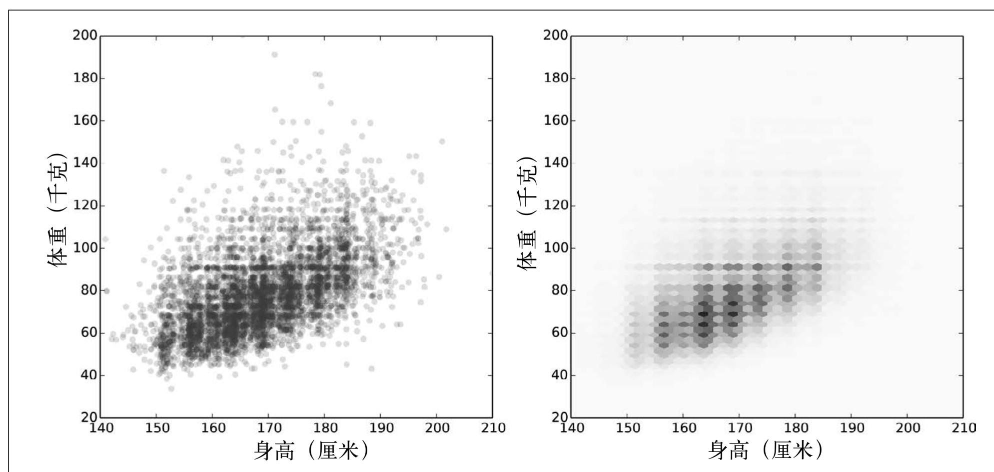



图 7-2：经过抖动和透明处理的散点图（左）和 hexbin 图（右）

对于中等规模的数据集，在散点图中设置透明度效果很好。BRFSS 数据有 414 509 条，图 7-2 只显示了其中的前 5000 条。

要处理规模更大的数据集，可以使用 hexbin 图。hexbin 图将图像划分为六角形的区间，将每个区间按照其中数据点的数量进行着色。thinkplot 提供 HexBin 方法。

```
thinkplot.HexBin(heights, weights)
```

图 7-2（右）展示了 hexBin 绘制结果。hexbin 的优点是可以很好地展示变量关系的形状，并且对于大数据集运行效率（时间效率和生成的文件大小）很高。缺点是离群值在图中不可见。

从这个示例我们可以看出，要绘制一个既能清晰展示变量关系，又不产生误导效果的散点图，并非易事。

## 7.2 描述关系特征

散点图能让我们对变量关系有个大体了解，而其他可视化方法则可以让我们更深入地了解变量关系的本质。一种方法是对一个变量进行分区，绘制另一个变量的百分位数。

NumPy 和 pandas 都提供数据分区函数。

```
df = df.dropna(subset=['htm3', 'wtkg2'])
bins = np.arange(135, 210, 5)
indices = np.digitize(df.htm3, bins)
groups = df.groupby(indices)
```

`dropna` 去除指定列含有 `nan` 值的数据行。`arrange` 产生一个 NumPy 区间数组，范围为从 135（不含）到 210，增量为 5。

`digitize` 计算出 `df.htm3` 中每个值所属区间的索引，结果为一个包含整数索引值的 NumPy 数组。如果值小于最低区间，则索引为 0；如果值大于最高区间，则索引为 `len(bins)`。

`groupby` 是一种 DataFrame 方法，返回一个 GroupBy 对象。当用在 `for` 循环中时，`groups` 遍历其中的组名，以及代表各组的 DataFrame 对象。因此，我们可以打印出每组中的行数。

```
for i, group in groups:
    print(i, len(group))
```

然后，对每个组，我们可以计算其身高均值和体重 CDF。

```
heights = [group.htm3.mean() for i, group in groups]
cdfs = [thinkstats2.Cdf(group.wtkg2) for i, group in groups]
```

最后，我们可以绘制身高对应的体重百分位数。

```
for percent in [75, 50, 25]:
    weights = [cdf.Percentile(percent) for cdf in cdfs]
    label = '%dth' % percent
    thinkplot.Plot(heights, weights, label=label)
```

图 7-3 展示了绘制结果。在 140~200 厘米，变量关系几乎是线性的。140~200 厘米这个范围涵盖了超过 99% 的数据，因此我们无需对极端值过多考虑。

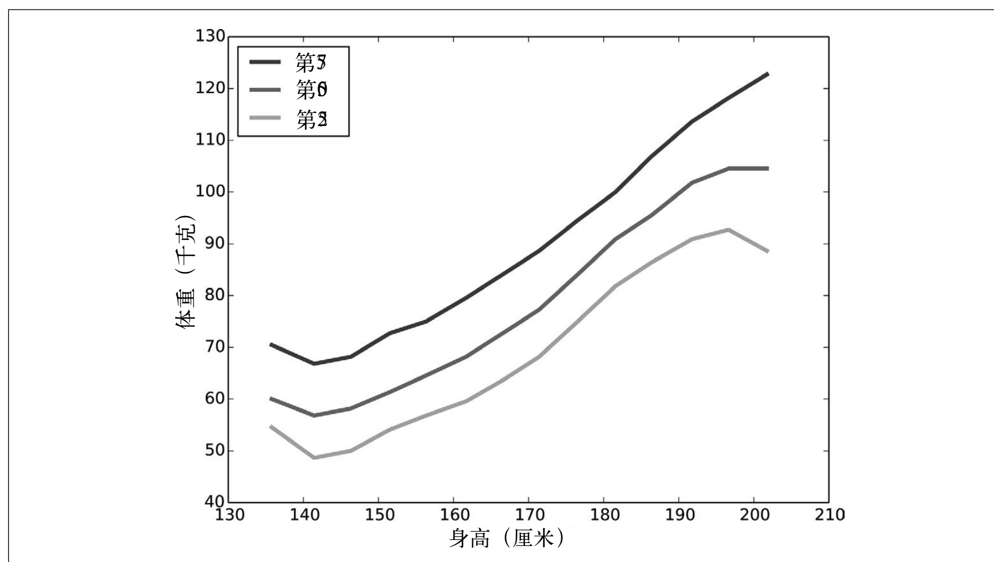



图 7-3：一组高度分区的体重百分位数

## 7.3 相关性

相关性（correlation）是一个统计量，用于量化两个变量之间关系的强弱。

度量相关性的困难之处在于，我们需要比较的变量通常使用不同的单位。即便变量使用相同的单位，也可能来自不同的分布。

这些问题有两个常见的解决方法。

- 将每个值都转换为标准分数（standard score），即其偏离均值的标准差数。这种转换会产生“Pearson 乘积矩相关系数”。
- 将每个值都转换为秩，即其在所有值的排序列表中的索引。这种转换会产生“Spearman 秩相关系数”。

假设 $X$ 由 $n$ 个值（$x_i$）构成，将每个值减去均值，然后除以标准差，即可得到标准分数：$z_i=(x_i-\mu)/\sigma$。

公式中的分子是一个偏差，即到均值的距离。除以 $\sigma$ 可以将偏差标准化（standardize），因此 $Z$ 的值是无量纲（无单位）的，其分布的均值为 0，方差为 1。

如果 $X$ 符合正态分布，那么 $Z$ 也符合正态分布。但如果 $X$ 是偏斜的或包含离群值，那么 $Z$ 也会偏斜或包含离群值。在这种情况下，使用百分位秩的稳健性更好。如果计算一个新变量 $R$，使 $r_i$ 为 $x_i$ 的秩，那么无论 $X$ 的分布如何，$R$ 总是从 1 到 $n$ 的均匀分布。

## 7.4 协方差

协方差（covariance）可以度量两个变量共同变化的趋势。如果我们有两个序列 $X$ 和 $Y$，那么序列中的值与均值的偏差分别为：

$$dx_i = x_i - \bar{x}$$

$$dy_i = y_i - \bar{y}$$

其中 $\bar{x}$ 是 $X$ 的样本均值，$\bar{y}$ 是 $Y$ 的样本均值。如果 $X$ 和 $Y$ 共同变化，那么这些偏差的正负性也会相同。

将这两组偏差相乘，如果偏差的正负性相同，那么乘积就是正数；如果偏差的正负性相反，那么乘积就是负数。因此，将这些乘积累加，就可以度量两组值共同变化的趋势。

协方差是这些乘积的均值。

$$\mathrm{Cov}(X, Y) = \frac{1}{n} \sum dx_i \, dy_i$$

其中 $n$ 为这两个序列的长度（两个序列的长度必须相等）。

如果你学习过线性代数，可能会发现 `Cov` 是两组偏差的点乘积除以其长度。因此，如果两个向量相同，则协方差值最大；如果两个向量正交，则协方差为 0；如果两个向量方向相反，则协方差为负数。`thinkstats2` 使用 `np.dot` 有效地实现了 `Cov` 算法。

```
def Cov(xs, ys, meanx=None, meany=None):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    if meanx is None:
        meanx = np.mean(xs)
    if meany is None:
        meany = np.mean(ys)

    cov = np.dot(xs-meanx, ys-meany) / len(xs)
    return cov
```

默认情况下，`Cov` 计算各点到样本均值的偏差，或者你也可以提供已知的均值。如果 `xs` 和 `ys` 是 Python 序列，那么 `np.asarray` 会将 `xs` 和 `ys` 转换为 NumPy 数组。如果 `xs` 和 `ys` 本身就是 NumPy 数据，那么 `np.asarray` 就不进行处理。

协方差的这个实现版本非常简单，易于解释。NumPy 和 pandas 也提供协方差的实现，但都用到了对小样本的矫正，而我们还未介绍过这部分知识。而且，`np.cov` 返回的是协方差矩阵，对我们目前而言过于复杂了。

## 7.5 Pearson相关性

协方差在一些计算中非常有用，但其含义很难解释，因此人们很少将协方差作为摘要统计量。别的不提，协方差的单位是 $X$ 和 $Y$ 的单位乘积，这一点就很难理解。例如，BRFSS 数据集中体重和身高的协方差是 113 千克－厘米，天晓得这是什么意思。

解决这个问题的方法之一是将偏差除以标准差，得到标准分数，然后计算标准分数的乘积：

$$p_i = \frac{(x_i - \bar{x})}{S_X} \frac{(y_i - \bar{y})}{S_Y}$$

其中 $S_X$ 和 $S_Y$ 分别是 $X$ 和 $Y$ 的标准差。这些乘积的均值为：

$$\rho = \frac{1}{n} \sum p_i$$

或者，我们可以通过分解 $S_X$ 和 $S_Y$ 改写标准差：

$$\rho = \frac{\mathrm{Cov}(X, Y)}{S_X S_Y}$$

这个公式以一位很有影响力的统计学家 Karl Pearson 的名字命名，称为 Pearson 相关性（Pearson's correlation）。Pearson 相关性容易计算，也易于解释。因为标准分数是无量纲（无单位），所以 $\rho$ 也是无单位的。

thinkstats2 中的 Pearson 相关性实现代码如下：

```
def Corr(xs, ys):
    xs = np.asarray(xs)
    ys = np.asarray(ys)

    meanx, varx = MeanVar(xs)
    meany, vary = MeanVar(ys)

    corr = Cov(xs, ys, meanx, meany) / math.sqrt(varx * vary)
    return corr
```

比起单独调用 np.mean 和 np.var，MeanVar 计算均值和协方差的性能稍好。

Pearson 相关性取值介于 −1~+1 之间（包含端点）。如果 $\rho$ 是正数，那么我们可以说两个变量正相关，即当一个变量值较大时，另一个变量的值也会较大。如果 $\rho$ 是负数，那么变量负相关，因此当一个变量值较大时，另一个变量值较小。

$\rho$ 的大小表明了相关性的强弱程度。如果 $\rho$ 为 1 或 −1，两个变量完全相关，也就是说，如

果你知道一个变量的值，就可以对另一个变量的值进行准确的预测。

现实世界中的大部分相关性都不是完全的，但相关性依然有其作用。身高和体重的相关性为 0.51，比起其他与人类相关的变量，身高和体重具有很强的相关性。

## 7.6 非线性关系

如果 Pearson 相关性接近 0，你可能会认为变量之间没有关系，但这个结论并不成立。Pearson 相关性只度量了线性（linear）关系。如果变量之间存在非线性关系，那么 $\rho$ 对变量相关性强弱的估计就可能是错误的。

图 7-4 摘取自 http://wikipedia.org/wiki/Correlation_and_dependence，展示了几个精心设计的数据集的散点图和相关系数。

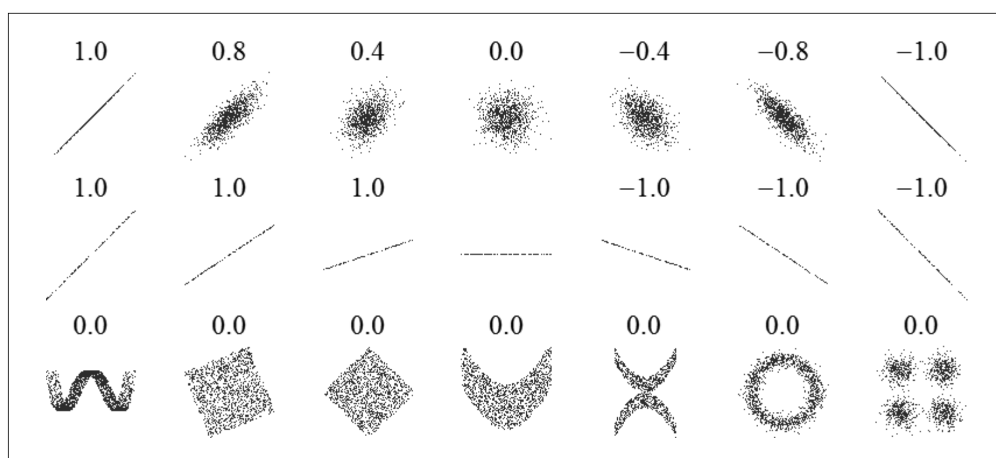



图 7-4：各种相关性的数据集示例

第 1 行展示了具有不同线性相关性的数据集。你可以从这一行直观感受不同 $\rho$ 值的相关性是怎样的。第 2 行展示了具有不同斜度的完全相关，说明相关性与斜度无关（我们很快会介绍如何估计斜度）。第 3 行展示的变量明显具有相关性，但因为变量之间的关系是非线性的，因此协方差为 0。

因此，在盲目计算协方差之前，我们首先应该查看数据的散点图，以免得出错误的结论。

## 7.7 Spearman秩相关

如果变量之间的关系是线性的，而且变量大致符合正态分布，那么 Pearson 相关性能够很好地说明相关性的强弱。但是离群值会影响 Pearson 相关性的稳健性。Spearman 秩相关能

够缓解离群值以及偏斜分布的的影响，也可以用于描述变量的相关性。要计算 Spearman 相关性，必须计算每个值的秩（rank），即该值在排序样本中的索引。例如，在样本 [1, 2, 5, 7] 中，值 5 出现在排序列表的第 3 位，因此秩为 3。然后我们需要计算这些秩的 Pearson 相关性。

thinkstats2 提供了一个计算 Spearman 秩相关的函数。

```
def SpearmanCorr(xs, ys):
    xranks = pandas.Series(xs).rank()
    yranks = pandas.Series(ys).rank()
    return Corr(xranks, yranks)
```

可以将函数的参数转换为 pandas Series 对象，从而使用 Series 提供的 rank 方法计算每个值的秩，结果返回一个 Series 对象，然后使用 Corr 计算这些秩的相关性。

也可以直接使用 Series.corr 方法，在参数中指定使用 Spearman 方法。

```
def SpearmanCorr(xs, ys):
    xranks = pandas.Series(xs)
    yranks = pandas.Series(ys)
    return xs.corr(ys, method='spearman')
```

BRFSS 数据的 Spearman 秩相关系数为 0.54，比 Pearson 相关性 0.51 略高。导致差值产生的原因有：

- 如果变量之间的关系是非线性的，那么 Pearson 相关性会低估相关性的强弱；
- 如果所研究的分布之一是偏斜的或包含离群值，那么 Pearson 相关性在两个方向都可能受到影响，而 Spearman 秩相关的稳健性较好。

在前面讨论的 BRFSS 数据示例中，我们知道体重数据大致符合对数正态分布。体重数据经过对数转换后，大致符合正态分布，数据分布不偏斜。因此，要消除数据分布偏斜的影响，还有一个方法是计算体重对数值和身高值的 Pearson 相关性。

```
thinkstats2.Corr(df.htm3, np.log(df.wtkg2)))
```

计算结果为 0.53，非常接近 Spearman 秩相关系数 0.54。这说明 Pearson 相关性和 Spearman 秩相关系数的差值主要是由体重分布的偏斜导致的。

## 7.8 相关性和因果关系

如果变量 A 和 B 相关，那么有 3 种可能的解释：A 导致 B，或 B 导致 A，或其他某些因素导致 A 和 B。这些解释称为“因果关系”（causal relationship）。

相关性本身并不能区分这些情况，因此无法告诉你哪种解释是正确的。这条规则通常可用

一句话进行总结："相关性并不意味着因果关系"，这句话一语中的，简明扼要，甚至有专门的维基百科页面：http://wikipedia.org/wiki/Correlation_does_not_imply_causation。

那么，如何才能证明因果关系的存在呢？

- 时间

如果 A 在 B 之前发生，那么 A 可能导致 B，而 B 不可能导致 A（至少按照我们对因果关系的理解是这样的）。事件的发生顺序可以帮助我们推导出谁是因谁是果，但并不能排除其他因素可能既导致了 A 又导致了 B。

- 随机性

如果将一个大型样本随机分为两组，计算任意变量的均值，那么两组结果的差别应该很小。如果两组的所有变量均值都相同，只有一个不同，那么你可以据此排除虚假关系的可能性。即使你不知道相关的变量是什么，也可以使用这种方法。但是如果你知道相关的变量是什么，就可以检查这些分组在各方面是相同的，那么效果将会更好。

正是这些想法催生了*随机对照试验*（randomized controlled trial）。在随机对照试验中，试验对象被随机分配到两个（或更多）组：*试验组*（treatment group）和*对照组*（control group）。试验组接受某些干预，例如新药物；对照组不接受干预，或者接受其他效果已知的干预。

随机对照试验是展示因果关系的最可靠方法，奠定了基于科学的医学研究（参见 http://wikipedia.org/wiki/Randomized_controlled_trial）。

不幸的是，对照试验只能在实验室科学、医学以及其他少数领域中进行。在社会科学中，对照试验通常无法进行或存在道德争议，因此非常少见。

揭示因果关系的另一个方法是寻找*自然实验*（natural experiment），对各方面相似的组实施不同的"处理"。自然实验存在一个危险，即实验组可能存在一些不易被发现的差异。你可以从 http://wikipedia.org/wiki/Natural_experiment 得到更多相关信息。

在某些情况下，我还可以使用*回归分析*（regression analysis）来推导出因果关系。第 11 章将详细介绍回归分析。

## 7.9 练习

本章练习的参考答案位于 chap07soln.py 中。

- 练习 7.1

请使用全国家庭增长调查数据，绘制新生儿体重与母亲年龄关系的散点图；绘制新生儿体重与母亲年龄的百分位数；计算 Pearson 和 Spearman 相关性。你如何描述这些变量之间的关系？

## 7.10 术语

- 散点图（scatter plot）
  两个变量之间关系的可视化展现，将每行数据显示为一个点。
- 抖动（jitter）
  为了可视化而在数据中加入的随机噪音。
- 饱和（saturation）
  多个点重叠而导致的信息丢失。
- 相关性（correlation）
  衡量两个变量之间关系强弱的统计量。
- 标准化（standardize）
  将一组值进行转换，使其均值为 0，方差为 1。
- 标准分数（standard score）
  一个标准化的值，表示为距离均值的标准差数。
- 协方差（covariance）
  对两个变量共同变化趋势的度量。
- 秩（rank）
  一个元素出现在排序列表中的索引。
- 随机对照试验（randomized controlled trial）
  一种实验设计，研究对象随机分组，不同的组接受不同的试验处理。
- 试验组（treatment group）
  对照试验中接受某些干预的组。
- 对照组（control group）
  对照试验中不接受干预的组，或者接受效果已知的干预。
- 自然实验（natural experiment）
  一种实验设计，特征是实验对象可以自然划分为至少近似随机的组。

# 第 8 章

# 估计

本章代码位于 estimation.py 中。前言介绍了如何下载和使用本书代码。

## 8.1 估计游戏

让我们来做一个游戏。我想一个分布，你来猜这个分布是什么。两个提示：这是一个正态分布；分布的随机样本如下。

```
[-0.441, 1.774, -0.101, -1.138, 2.975, -2.138]
```

你猜这个分布的均值 $\mu$ 是多少？

一个方法是使用样本均值 $\bar{x}$ 作为 $\mu$ 的估计。在本例中，$\bar{x}$ 为 0.155，因此合理的猜测是 $\mu$=0.155。这个过程称为*估计*（estimation），我们使用的统计量（样本均值）称为*估计量*（estimator）。

使用样本均值估计 $\mu$ 是显而易见的方法，我们似乎很难想出其他方法。但是，假设我们对游戏稍作修改，引入离群值。

*我在想一个分布。*这是个正态分布，一位调查员收集了这个分布的一个样本。这位调查员工作不太认真，有时会把小数点写错位置，他收集的样本如下：

```
[-0.441, 1.774, -0.101, -1.138, 2.975, -213.8]
```

现在你该如何估计 $\mu$？如果使用样本均值，你的估计值会是 −35.12。使用样本均值进行估计是最好的方法吗？还有其他估计方法吗？

一种方法是找出并去除离群值，然后计算剩余的样本均值。另一种方法是使用中位数作为估计量。

使用哪种估计量效果更好？这取决于具体情况（例如是否存在离群值）及目标。你要使误差最小，还是要使答案正确的几率最大？

如果没有离群值，那么使用样本均值作为估计量能够使*均方误差*（mean squared error，MSE）最小。也就是说，如果我们将这个估计游戏进行多次，计算每次的误差$\bar{x}-\mu$，那么样本均值可以使下面的值最小：

$$\mathrm{MSE}=\frac{1}{m}\sum(\bar{x}-\mu)^2$$

其中 $m$ 为游戏次数，请不要把它和 $n$ 搞混了，$n$ 是用于计算$\bar{x}$的样本大小。

下面的函数模拟了这个估计游戏，计算均方误差的平方根，即*均方根误差*（root mean squared error，RMSE）。

```
def Estimate1(n=7, m=1000):
    mu = 0
    sigma = 1

    means = []
    medians = []
    for _ in range(m):
        xs = [random.gauss(mu, sigma) for i in range(n)]
        xbar = np.mean(xs)
        median = np.median(xs)
        means.append(xbar)
        medians.append(median)

    print('rmse xbar', RMSE(means, mu))
    print('rmse median', RMSE(medians, mu))
```

再次提醒：`n` 是样本的大小，而 `m` 是游戏重复的次数。`means` 是基于$\bar{x}$的估计列表。`medians` 是中位数列表。

下面的函数计算均方根误差。

```
def RMSE(estimates, actual):
    e2 = [(estimate-actual)**2 for estimate in estimates]
    mse = np.mean(e2)
    return math.sqrt(mse)
```

`estimates` 是估计列表，`acutal` 是待估计的实际值。当然，在实际应用中我们无法知道 `actual`。如果 `actual` 已知，那我们就无需进行估计了。这个实验的目的是比较这两个估计量的性能优劣。

运行上面的代码，我们得到样本均值的均方根误差为 0.41。也就是说，如果我们基于 $n$=7 的样本，使用$\bar{x}$估计分布的均值，那么预期的偏差平均为 0.41。使用中位数估计均值得到的均方根误差为 0.53，证实使用$\bar{x}$的均方根误差较小，至少在本例中是如此。

使均方误差最小当然很好，但不一定是最佳策略。例如，假设我们要估计一个建筑工地的风速分布。如果估计的结果太高，那么我们可能会不必要地增加结构强度，导致成本增加；但如果估计的结果太低，建筑物可能会倒塌。估计值偏大或是偏小，导致的成本变化并不相同，因此一味追求均方误差最小并非最佳策略。

再举个例子，假设我掷 3 次骰子，让你预测点数总和。如果你猜对了，就会赢得奖品，猜错了则空手而归。在这种情况下，使均方误差最小的估计值是 10.5，但是这个猜测显然不靠谱，因为掷 3 次骰子得到的总点数不可能是 10.5。此时，你希望作出最可能与实际值相符的估计，即*最大似然估计量*（maximum likelihood estimator，MLE）。如果你猜 10 或者 11，正确的可能性最大，为 1/8。

# 8.2 猜测方差

*我在想一个分布。*这是一个正态分布，样本（依旧）如下：

```
[-0.441, 1.774, -0.101, -1.138, 2.975, -2.138]
```

你猜这个分布的方差 $\sigma^2$ 是多少？最显而易见的办法是用样本方差 $S^2$ 作为估计量。

$$S^2 = \frac{1}{n}\sum(x_i - \bar{x})^2$$

对于大样本，$S^2$ 是不错的估计量。但是对于小样本，$S^2$ 通常比方差分布低很多。由于这个糟糕的属性，人们将 $S^2$ 称为*偏倚*（biased）估计量。如果对于多次重复实验，一个估计量的预期误差总和（或均值）为 0，那么这个估计量就是*无偏的*（unbiased）。

幸好，还有一个简单统计量是 $\sigma^2$ 的无偏估计量。

$$S_{n-1}^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2$$

至于为什么 $S^2$ 是偏倚的，而$S_{n-1}^2$是无偏的，请参考 http://wikipedia.org/wiki/Bias_of_an_estimator。

这个估计量的最大问题是，名字和符号用法不一致：名字是“样本方差”，既可以指 $S^2$，也可以指$S_{n-1}^2$，而二者都使用符号 $S^2$。

下面的函数模拟了这个估计游戏，并测试 $S^2$ 和$S_{n-1}^2$的性能。

```
def Estimate2(n=7, m=1000):
    mu = 0
    sigma = 1

    estimates1 = []
    estimates2 = []
    for _ in range(m):
        xs = [random.gauss(mu, sigma) for i in range(n)]
        biased = np.var(xs)
        unbiased = np.var(xs, ddof=1)
        estimates1.append(biased)
        estimates2.append(unbiased)

    print('mean error biased', MeanError(estimates1, sigma**2))
    print('mean error unbiased', MeanError(estimates2, sigma**2))
```

再次提醒：`n` 是样本大小，而 `m` 是游戏进行的次数。默认情况下，`np.var` 计算 $S^2$；如果指定参数 `ddof=1`，`ddof` 代表“自由度增量”（delta degrees of freedom），`np.var` 则计算$S_{n-1}^2$。我在这里不会解释自由度增量是什么，你可以参考 http://en.wikipedia.org/wiki/Degrees_of_freedom_(statistics)。

`MeanError` 计算估计值和实际值之间差别的均值。

```
def MeanError(estimates, actual):
    errors = [estimate-actual for estimate in estimates]
    return np.mean(errors)
```

运行上面的代码，得到 $S^2$ 的误差均值为 −0.13。正如我们所料，这个偏倚估计量总是偏低。$S_{n-1}^2$的误差均值为 0.014，约为 $S^2$ 误差均值的 1/10。随着 `m` 增加，我们预期$S_{n-1}^2$的误差均值会逐渐接近 0。

均方误差和偏倚都属于长期属性，建立在估计游戏多次重复的基础上。通过运行如本章所示的模拟代码，我们可以比较估计量，检验其是否具有我们预期的属性。

但是，如果将一个估计量用于真实数据，你只能进行一次估计。在这种情况下，说一个估计无偏没有任何意义。无偏是估计量的属性，而不是某次估计的属性。

选择具有适当属性的估计量，并用其生成一个估计后，下一步是描述这个估计的不确定性。这就是下一节讨论的内容。

## 8.3　抽样分布

假设你是一位科学家，在野生动物保护区研究大猩猩。你想知道保护区中成年雌性大猩猩的平均体重。要给猩猩称重，你必须将它们麻醉。这种做法既危险又有很高成本，还可能

伤害大猩猩。但是，如果获得保护区中成年雌性大猩猩的平均体重非常重要，我们也许可以测量 9 只大猩猩的体重作为样本。假设我们对保护区中大猩猩的情况非常了解，那么就能选出具有代表性的成年雌性大猩猩样本。我们可以使用这个样本均值 $\bar{x}$ 来估计未知的总体均值 $\mu$。

测量了 9 只雌性大猩猩的体重后，你得到 $\bar{x} = 90$ 千克，样本均方差 $S = 7.5$ 千克。样本均值是总体均值 $\mu$ 的无偏估计量，在多次重复实验中可以使均方误差最小。因此，如果需要用一个估计值来概括结果，你应该估计保护区中成年雌性大猩猩的平均体重为 90 千克。

但是，你对这个估计值的准确性有多少信心呢？如果只是从数量庞大的大猩猩总体中抽取 9 只测量体重，有可能运气不好，抽到了最重的 9 只（或者最轻的 9 只）。由随机选择导致的估计变化称为*抽样误差*（sampling error）。

为了量化抽样误差，我们可以假定 $\mu$ 和 $\sigma$ 的取值，模拟抽样过程，观察 $\bar{x}$ 如何变化。

由于总体 $\mu$ 和 $\sigma$ 的实际值未知，因此我们将使用估计值 $\bar{x}$ 和 $S$。我们需要回答的问题是：如果 $\mu$ 和 $\sigma$ 的实际值分别为 90 千克和 7.5 千克，在多次运行相同的实验后，估计均值 $\bar{x}$ 将如何变化？

解答这个问题的代码如下：

```
def SimulateSample(mu=90, sigma=7.5, n=9, m=1000):
    means = []
    for j in range(m):
        xs = np.random.normal(mu, sigma, n)
        xbar = np.mean(xs)
        means.append(xbar)

    cdf = thinkstats2.MakeCdfFromList(means)
    ci = cdf.Percentile(5), cdf.Percentile(95)
    stderr = RMSE(means, mu)
```

`mu` 和 `sigma` 是参数的*假定*值。`n` 是样本大小，即我们称重的大猩猩数。`m` 是运行模拟的次数。

在每次循环中，我们从具有给定参数的正态分布中选择 `n` 个值，计算样本均值 `xbar`。代码进行 1000 次模拟过程，然后计算估计值分布的 `cdf`。结果如图 8-1 所示。这个分布称为估计量的*抽样分布*（sampling distribution），展示了多次重复实验时估计值的变化。

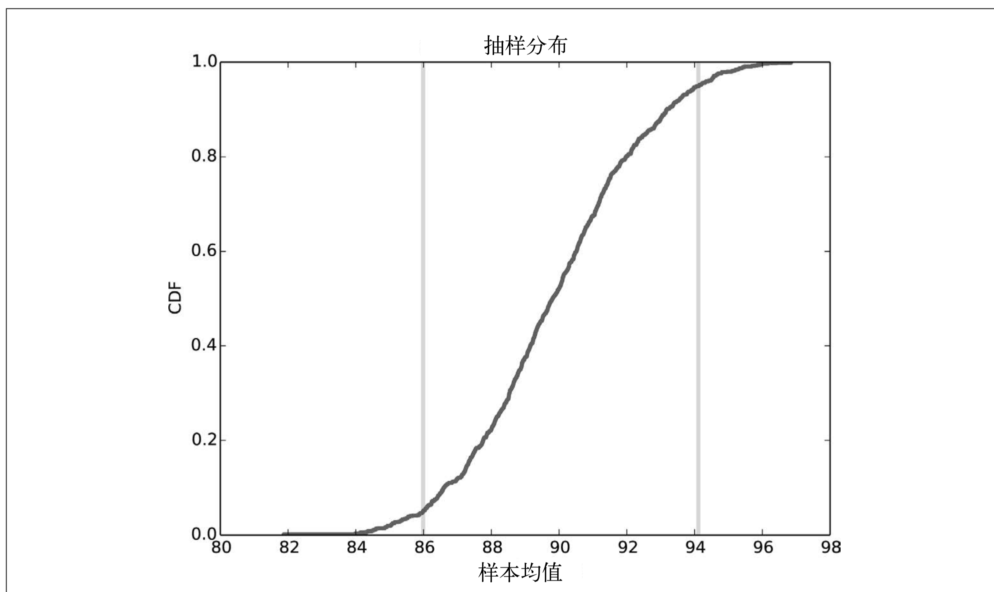



图 8-1：$\bar{x}$的抽样分布以及置信区间

抽样分布的均值与假定值 $\mu$ 非常接近，说明平均而言，这个实验产生了正确的结果。在 1000 次重复实验中，最低结果为 82 千克，最高结果为 98 千克。这个范围说明估计值与实际值的偏差可能达到 8 千克。

人们通常用两种方法对抽样分布进行概括。

标准误差（standard error，SE）

度量预期估计值平均偏离实际值多少。对每次模拟实验，先计算误差$\bar{x} - \mu$，然后计算均方根误差，即得到标准误差。在这个示例中，标准误差约为 2.5 千克。

置信区间（confidential interval，CI）

包含抽样分布中指定比例的范围。例如，90% 置信区间是从第 5 百分位数到第 95 百分位数。在这个示例中，90% CI 是 (86,94) 千克。

关于标准误差和置信区间的误解有很多。

- 人们常常混淆标准误差和标准差。请记住：标准差描述的是度量值的变化。在这个示例中，大猩猩体重的标准差是 7.5 千克。标准误差描述的是估计值的变化。在这个示例中，基于 9 个测量样本估计出的均值标准误差为 2.5 千克。

  标准误差和标准差有一个区别：随着样本量的增加，标准误差会变小，而标准差则不变。
- 人们常常认为，实际参数 $\mu$ 落入 90% 置信区间的概率为 90%。不幸的是，这种想法并不正确。如果你要得出类似的结论，就必须使用贝叶斯方法（请参考我写的 *Think*

*Bayes* 一书）。

抽样分布回答的是另一个问题，告诉你重复进行实验时一个估计值会如何变化，使你对估计值的可靠性有所了解。

置信区间和标准误差只量化了抽样误差，即由于只测量了总体中的部分成员而导致的误差。记住这一点非常重要。抽样分布不考虑其他情况的误差，特别是抽样偏倚和测量误差。下一节将讨论抽样偏倚和测量误差。

## 8.4 抽样偏倚

假设你不需要测量自然保护区中大猩猩的体重，而是要了解所在城市女性的平均体重。此时你不太可能选取一个具有代表性的样本，并测量她们的体重。

一个简单的方法是“电话抽样”：你可以从电话簿中选取随机号码，致电要求与一位成年女性通话，并询问她的体重。

电话抽样具有明显的局限性。例如，样本限于公开了电话号码的居民，因而排除了没有电话的人（这些人可能比较穷）和未公布号码的人（这些人可能比较富有）。而且，如果你在白天拨打家庭电话，就不太可能访问到上班的人。如果你只对接电话的人进行访问，就不太可能访问到与别人共享一个号码的人。

如果收入、就业状况和家庭人口等因素与体重相关（确实可能），那么你的调查结果就会受到这样或那样的影响。这种问题是抽样过程的属性，因此称为*抽样偏倚*（sampling bias）。

抽样过程也可能受到个人意愿的影响。这也是一种抽样偏倚。有些人可能拒绝回答调查问题，如果拒绝回答问题的倾向与体重有关，也会影响调查的结果。

最后，如果你只是询问别人的体重，而不进行测量，那么得到的结果可能是不准确的。即便调查参与者很愿意回答你的问题，但如果她们对自己的真实体重不满意，也有可能会对数字进行一些“美化”，更何况并不是所有的参与者都那么配合。这些都属于*测量误差*（measurement error）。

在汇报一个估计值时，你可以给出标准误差或置信区间，也可以都给出以量化抽样误差。但是，你要记住：抽样误差只是误差的来源之一，而且通常并不是最大的误差来源。

## 8.5 指数分布

让我们再玩一次猜测游戏吧。*我在想一个分布*。这是一个指数分布，样本如下：

```
[5.384, 4.493, 19.198, 2.790, 6.122, 12.844]
```

你猜这个分布的参数 $\lambda$ 是多少？

通常，指数分布的均值是 $1/\lambda$，那么倒推一下，我们应该选择：

$$L = 1/\bar{x}$$

$L$ 是 $\lambda$ 的一个估计量。这个估计量很特别，它还是最大似然估计量（参见 http://wikipedia.org/wiki/Exponential_distribution#Maximum_likelihood）。因此，如果你想最大化自己猜中 $\lambda$ 的几率，就应该选择 $L$。

但是，我们知道当存在离群值时，$\bar{x}$的健壮性不佳，因此预期 $L$ 也具有同样的问题。

我们可以选择基于样本中位数的估计量。指数分布的中位数是 $\ln(2)/\lambda$，倒推一下，我们可以定义一个估计量：

$$L_m=\ln(2)/m$$

其中 $m$ 是样本的中位数。

为了测试这两个估计量的性能，我们可以对抽样过程进行模拟。

```
def Estimate3(n=7, m=1000):
    lam = 2

    means = []
    medians = []
    for _ in range(m):
        xs = np.random.exponential(1.0/lam, n)
        L = 1 / np.mean(xs)
        Lm = math.log(2) / thinkstats2.Median(xs)
        means.append(L)
        medians.append(Lm)

    print('rmse L', RMSE(means, lam))
    print('rmse Lm', RMSE(medians, lam))
    print('mean error L', MeanError(means, lam))
    print('mean error Lm', MeanError(medians, lam))
```

以 $\lambda = 2$ 运行这个实验，得到的 $L$ 的均方根误差为 1.1。对于基于中位数的估计量 $L_m$，均方根误差为 1.8。从这个实验中，我们无法得知 $L$ 是否使均方误差最小，但是至少看起来 $L$ 比 $L_m$ 更接近真实值。

不幸的是，这两个估计量似乎都是偏倚的。$L$ 的均值误差为 0.33，$L_m$ 的均值误差为 0.45，而且随着样本量 `m` 的增加，二者的均值误差都不会趋近于 0。

实际上，$\bar{x}$是指数分布均值 $1/\lambda$ 的无偏估计量，而 $L$ 却不是 $\lambda$ 的无偏估计量。

## 8.6 练习

你可以复制 estimation.py，以此副本为基础进行接下来的练习。本章练习的参考答案位于 chap08soln.py 中。

- 练习 8.1

本章，我们使用$\bar{x}$和中位数估计 $\mu$，并发现$\bar{x}$的均方误差较小。而且，我们还使用了 $S^2$ 和 $S_{n-1}^2$估计 $\sigma$，发现 $S^2$ 是 $\sigma$ 的偏倚估计量，而$S_{n-1}^2$是无偏的。

请进行类似的实验，检验$\bar{x}$和中位数是否是 $\mu$ 的偏倚估计，并检查 $S^2$ 和$S_{n-1}^2$中哪个的均方误差较小。

- 练习 8.2

假设你从一个 $\lambda$=2 的指数分布中抽取了 $n$=10 的样本。请将这个实验模拟 1000 次，绘制估计值 $L$ 的抽样分布。请计算这个估计值的标准误差及其 90% 置信区间。

请使用几个不同的 $n$ 值重复这个实验，绘制标准误差与 $n$ 的关系图。

- 练习 8.3

在冰球和足球比赛中，进球的时间间隔大致符合指数分布。因此，你可以通过观察一个球队在比赛中的进球次数来估计进球得分率。这个估计过程与对进球时间间隔进行抽样略有不同，让我们看看具体是如何进行的。

请编写一个函数模拟一场比赛的过程，以平均每场比赛的得分率 `lam` 为参数，生成进球的时间间隔，直到总时间超过一场比赛的时长，然后返回得分数。

请再编写一个函数，模拟多场比赛，保存 `lam` 的估计值，然后计算这些值的均值误差和均方根误差。

这种估计方法是偏倚的吗？请绘制这个估计值的抽样分布和 90% 置信区间。这个估计量的标准误差是多少？如果 `lam` 的值增加，抽样误差会发生何种变化？

## 8.7 术语

- 估计（estimation）
  从样本推导出分布参数的过程。

- 估计量（estimator）
  用于估计一个参数的统计量。

- 均方误差（mean squared error，MSE）
  对估计误差的度量。

- 均方根误差（root mean squared error，RMSE）
  均方误差的平方根，能够更有意义地表示典型误差的量级。
- 最大似然估计量（maximum likelihood estimator，MLE）
  一种估计量，计算最有可能正确的点估计。
- 估计量偏倚（bias of an estimator）
  在重复实验中，一个估计量高于或低于参数实际值的平均趋势。
- 抽样误差（sampling error）
  因为样本规模有限以及随机变化而导致的估计错误。
- 抽样偏倚（sampling bias）
  因为抽样过程产生的样本不能代表总体而导致的估计误差。
- 测量误差（measurement error）
  因为收集或记录数据不准确而导致的估计误差。
- 抽样分布（sampling distribution）
  实验重复多次得到的统计量分布。
- 标准误差（standard error）
  一个估计的均方根误差，对抽样误差（而非其他误差源）导致的波动进行量化。
- 置信区间（confidence interval）
  一个区间，代表实验重复多次时预期的估计量范围。

# 第 9 章

# 假设检验

本章代码位于 hypothesis.py 中。前言介绍了如何下载和使用本书代码。

## 9.1 经典假设检验

在探索 NSFG 数据的过程中，我们看到了几个“直观效应”，其中包括第一胎和其他胎的区别。到目前为止，我们看到的只是表面现象，本章则将对这些效应进行检验。

我们想解答的基本问题是，在一个样本中观察到的效应是否也会出现在更大规模的总体中。例如，在 NSFG 样本中，第一胎和其他胎的妊娠期长度不同，我们想了解这种效应是真实反映了美国妇女的生育情况，还是偶然出现在这个样本中而已。

这个问题有好几种表示方法，包括 Fisher 原假设检验、Neyman-Pearson 决策理论和贝叶斯推理[1]，大部分人在实践中使用的都是这 3 种方法。这里我要介绍的是这些方法的一个子集，称为经典假设检验（classical hypothesis testing）。

经典假设检验的目的是回答一个问题：“给定一个样本和一个直观效应，这个效应是偶然出现的概率为多少？”回答这个问题的步骤如下。

- 第一步，选择一个检验统计量（test statistic），对直观效应进行量化。在全国家庭增长调查统计示例中，直观效应是第一胎和其他胎的妊娠时间存在差异，因此我们很自然地选择这两个群组的均值差作为检验统计量。

注 1：要了解贝叶斯推理，请参考同系列的 *Think Bayes* 一书。

- 第二步，定义*原假设*（null hypothesis）。原假设是系统的一个模型，所基于的假设是直观效应不为真。在全国家庭增长调查统计示例中，原假设是第一胎和其他胎没有区别，即两个群组的妊娠时间具有相同的分布。
- 第三步，计算 $p$ 值（p-value）。$p$ 值是在原假设为真时，直观效应出现的概率。在全国家庭增长调查统计示例中，我们将计算两个群组均值的实际差异，然后计算在原假设为真的情况下，这个差异等于或大于实际值的概率。
- 最后，解释结果。如果 $p$ 值很低，那么我们称这个效应是*统计显著*（statistically significant）的，即不太可能偶然发生。在这种情况下，我们推断，这个效应在大规模总体中出现的可能性更大。

上述过程的逻辑与反证法类似。要证明一个数学陈述 A，你可以暂时假设 A 不为真，如果这个假设推导出了矛盾结果，那么就可以断定 A 确实为真。

同样，为了检验一个如“这个效应为真”的假设，我们暂时假设这个效应不为真，即原假设。基于这个假设，我们计算这个直观效应的概率，即 $p$ 值。如果 $p$ 值很低，我们就可以断定原假设不太可能为真。

## 9.2 假设检验

`thinkstats2` 提供一个类 `HypothesisTest`，表示一个经典假设检验结果，定义如下：

```
class HypothesisTest(object):

    def __init__(self, data):
        self.data = data
        self.MakeModel()
        self.actual = self.TestStatistic(data)

    def PValue(self, iters=1000):
        self.test_stats = [self.TestStatistic(self.RunModel())
                           for _ in range(iters)]

        count = sum(1 for x in self.test_stats if x >= self.actual)
        return count / iters

    def TestStatistic(self, data):
        raise UnimplementedMethodException()

    def MakeModel(self):
        pass

    def RunModel(self):
        raise UnimplementedMethodException()
```

`HypothesisTest` 是一个抽象父类，完整定义了一些方法，并为其他方法预留接口。基于 `HypothesisTest` 的子类继承了 `__init__` 和 `PValue`，实现了 `TestStatistic` 和 `RunModel`，可

选择是否定义 MakeModel。

__init__ 可以以任何适宜数据为参数。__init__ 调用 MakeModel 构建原假设，然后将数据传递给 TestStatistics，从而计算样本中的效应规模。

方法 PValue 计算原假设条件下直观效应的概率。PValue 的参数 iters 是运行模拟的次数。PValue 第一行生成模拟数据，计算检验统计量并存放在 test_stats 中，返回的结果是 test_stats 中等于或超过检验统计量的观测值 self.actual 的元素所占的比例。

举个简单的例子[1]，假设我们投掷一枚硬币 250 次，结果得到 140 次正面和 110 次反面。基于这个结果，我们怀疑这个硬币质地不均匀，落地时正面朝上的可能性更大。为了检验这个假设，我们计算出当硬币质地均匀时出现这种结果的概率。

```
class CoinTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        heads, tails = data
        test_stat = abs(heads - tails)
        return test_stat

    def RunModel(self):
        heads, tails = self.data
        n = heads + tails
        sample = [random.choice('HT') for _ in range(n)]
        hist = thinkstats2.Hist(sample)
        data = hist['H'], hist['T']
        return data
```

参数 data 是一对整数，即结果为正面和反面的次数。检验统计量是两者的差值，因此 self.actual 为 30。

RunModel 在假设硬币质地均匀的条件下模拟投掷硬币的实验。RunModel 生成包含 250 次投掷结果的样本，使用 Hist 计算结果中正面和反面的数量，返回一对整数值。

现在，我们只需要实例化 CoinTest 并调用 PValue 就可以了。

```
ct = CoinTest((140, 110))
pvalue = ct.PValue()
```

结果约为 0.07。也就是说，如果硬币是质地均匀的，我们预期有 7% 的可能性看到正面和反面的差值达到 30 的情况。

我们应该如何解释这一结果呢？按照惯例，5% 是统计显著的阈值。如果 $p$ 值小于 5%，那么我们认为该效应是显著的，否则不是。

注 1：改编自 MacKay 的 *Information Theory, Inference, and Learning Algorithms*，2003。

但是 5% 这个值是随意设定的，而且（我们稍后将看到）$p$ 值依赖于检测统计量的选择和原假设模型。因此，我们不应该将 $p$ 值看作一个精确的度量。

我建议你按照重要性顺序进行解释：如果 $p$ 值小于 1%，那么效应不太可能是随机产生的；如果 $p$ 值大于 10%，那么效应可以合理解释为随机现象。位于 1% 和 10% 之间的 $p$ 值应看作边缘值。因此，在这个示例中，我认为这些数据没有提供足够的证据，无法证明硬币是否质地均匀。

## 9.3 检验均值差

人们最常检验的效应之一是两组样本的均值差。在全国家庭增长调查数据中，我们看到第一胎比其他胎的妊娠时间稍长，出生体重略轻。现在，我们要看看这些效应是否统计显著。

这些例子的原假设是两组样本的分布相同。对这个原假设建模，一个方法是*置换*（permutation），即从两组中取值混排，把两个组当成一个大组。

```
class DiffMeansPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = abs(group1.mean() - group2.mean())
        return test_stat

    def MakeModel(self):
        group1, group2 = self.data
        self.n, self.m = len(group1), len(group2)
        self.pool = np.hstack((group1, group2))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data
```

`data` 是一对序列，每组一个序列。检验统计量是两组序列的均值差。

`MakeModel` 记录了两个组的大小 `n` 和 `m`，将两组合并形成一个 NumPy 数组 `self.pool`。

`RunModel` 将两组合并后的数值混排，分为大小为 `n` 和 `m` 的两个组，以此模拟原假设。按照惯例，`RunModel` 返回的值与观察值的格式相同。

为了测试妊娠时间的差异，我们运行如下代码：

```
live, firsts, others = first.MakeFrames()
data = firsts.prglngth.values, others.prglngth.values
ht = DiffMeansPermute(data)
pvalue = ht.PValue()
```

`MakeModel` 读取全国家庭增长调查数据，返回代表所有成功生产、第一胎及其他胎的 DataFrame。我们将妊娠时间数据抽取为 NumPy 数组，传递给 `DiffMeansPermute`，计算出 *p* 值。计算结果约为 0.17，即我们预期有 17% 的可能性看到妊娠时间差达到所观测的差值。因此，这个效应不是统计显著的。

`HypothesisTest` 提供方法 `PlotCdf`，绘制检验统计量的分布，并用一条灰线标识观察到的效应规模。

```
ht.PlotCdf()
thinkplot.Show(xlabel='test statistic',
               ylabel='CDF')
```

图 9-1 展示了绘制结果。CDF 与观察到的差值在 0.83（即 *p* 值 0.17 的补）处相交。

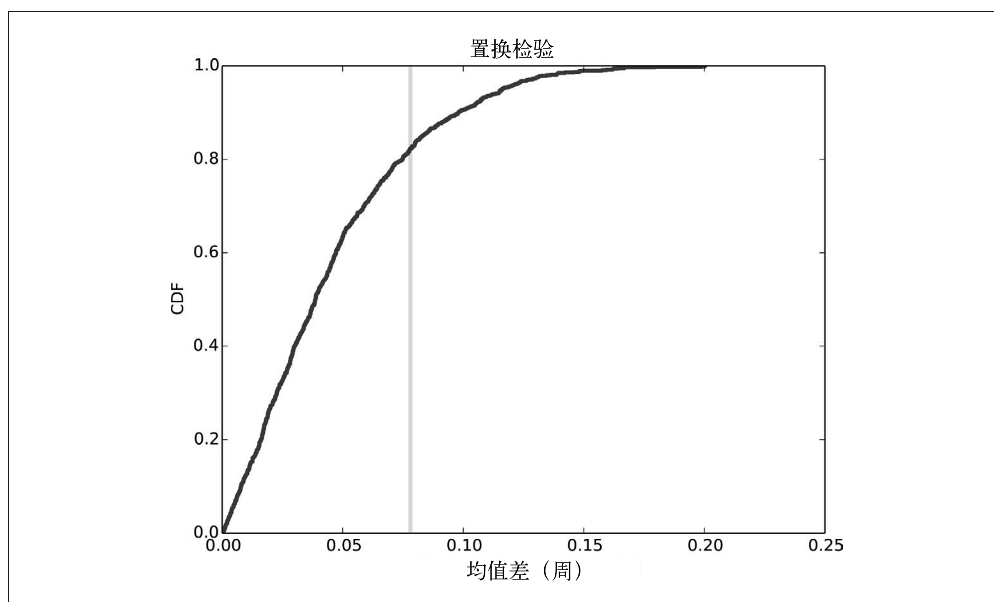



**图 9-1：原假设下，妊娠期时间均值差 CDF**

如果我们使用新生儿体重数据进行同样的分析，得到的 *p* 值为 0。运行 1000 次模拟，均值差都没有达到观察到的差值 0.12 磅。因此，我们将报告 $p<0.001$，得出的结论为，新生儿体重的差值是统计显著的。

## 9.4 其他检验统计量

你应当根据需要解决的实际问题选择检验统计量。例如，如果研究第一胎的妊娠时间是否与其他胎不同，那么检验均值差就是很好的选择，我们在前一节中正是这么做的。

如果我们有理由认为第一胎可能出生较晚，那么就不应该检验差值，而是使用另一个的检验统计量。

```
class DiffMeansOneSided(DiffMeansPermute):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = group1.mean() - group2.mean()
        return test_stat
```

`DiffMeansOneSided` 从父类 `DiffMeansPermute` 继承了 `MakeModel` 和 `RunModel` 方法，唯一的不同是 `TestStatistic` 方法计算的是组一均值减去组二均值，没有取绝对值。这种检验只检查差值分布的一侧，因此称为*单侧的*（one-sided）检验。之前的检验使用差值分布的两侧，因而称为*双侧的*（two-sided）检验。

这一版本测试得到的 $p$ 值为 0.09。单侧检验的 $p$ 值通常约为双侧检验 $p$ 值的一半，但会受分布形状的影响。

我们的单侧假设为第一胎出生较晚，这个假设比双侧假设更为具体，因此 $p$ 值较小。但即便对于这个更具体的假设，差值也不是统计显著的。

我们可以使用同样的框架检验标准差的差值。在 3.3 节中，有一些证据显示第一胎更可能提早或推迟出生，较少准时。因此，我们可以假设第一胎妊娠时间的标准差更高。检验方法如下：

```
class DiffStdPermute(DiffMeansPermute):

    def TestStatistic(self, data):
        group1, group2 = data
        test_stat = group1.std() - group2.std()
        return test_stat
```

我们的假设是第一胎的标准差更高，而不仅仅是不同，因此这个检验是单侧检验。检验得到的 $p$ 值为 0.09，不是统计显著的。

## 9.5　检验相关性

这个框架也可以检验相关性。例如，在全国家庭增长调查数据集中，新生儿体重和母亲年龄的相关性约为 0.07。年龄较大的母亲似乎产下的孩子更重。但是，这种效应是偶然产生的吗？

我选择 Pearson 相关性作为检验统计量，但 Spearman 相关性也是很好的选择。如果我们有理由预期正相关，那么就可以进行单侧检验。但由于我们并没有任何相关证据，因此还是选择使用相关性的绝对值进行双侧检验。

检验的原假设是母亲年龄和新生儿体重之间没有相关性。我们可以将观察值混排进行模拟，在这个模拟世界中，母亲年龄和新生儿体重的分布仍保持不变，但这两个变量之间没有相关性。

```
class CorrelationPermute(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        xs, ys = data
        test_stat = abs(thinkstats2.Corr(xs, ys))
        return test_stat

    def RunModel(self):
        xs, ys = self.data
        xs = np.random.permutation(xs)
        return xs, ys
```

`data` 是一对序列。`TestStatistic` 计算 Pearson 相关性的绝对值。`RunModel` 将 `xs` 进行混排，返回模拟数据。

下面一段代码读取数据并运行检验：

```
live, firsts, others = first.MakeFrames()
live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
data = live.agepreg.values, live.totalwgt_lb.values
ht = CorrelationPermute(data)
pvalue = ht.PValue()
```

我使用指定 `subset` 参数的 `dropna` 方法，去除缺失所需任一变量的数据行。

实际数据的相关性为 0.07。检验计算得到的 $p$ 值为 0。在 1000 次重复实验中，模拟得到的最大相关性为 0.04。因此，虽然观察到的变量相关性很小，但这种相关性是统计显著的。

这个示例提醒我们，“统计显著”并不一定说明一个效应是重要的，或者在实践中是显著的。“统计显著”只说明一个效应不太可能是偶然产生的。

## 9.6 检验比例

假设你经营一家赌场，怀疑一位顾客使用作弊骰子，也就是说这个骰子经过处理，更容易掷出其中一面。你抓住这位受怀疑的作弊者，没收了骰子，但是还必须证明这个骰子有问题。你将这个骰子掷了 60 次，得到如下结果：

| 点数 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 频数 | 8 | 9 | 19 | 5 | 8 | 11 |

你希望的结果是每个点数平均出现 10 次。在这个数据集中，3 出现的次数较多，4 较少。但是，这些差异是统计显著的吗？

为了检验这个假设，我们可以计算出每个值的预期频数、预期频数与观察频数的差值，以及差值绝对值的和。在这个示例中，我们预期 60 次投掷里，骰子的每一面都出现 10 次，观察值与预期值的差值为 −2、−1、9、−5、−2 和 1，差值绝对值的和为 20。完全偶然出现这么大差值的概率是多少呢？

下面这个版本的 HypothesisTest 回答了这个问题。

```
class DiceTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        observed = data
        n = sum(observed)
        expected = np.ones(6) * n / 6
        test_stat = sum(abs(observed - expected))
        return test_stat

    def RunModel(self):
        n = sum(self.data)
        values = [1, 2, 3, 4, 5, 6]
        rolls = np.random.choice(values, n, replace=True)
        hist = thinkstats2.Hist(rolls)
        freqs = hist.Freqs(values)
        return freqs
```

代码中使用的数据表示一列频数。观察值为 [8, 9, 19, 5, 8, 11]，预期频数都是 10。检验统计量是差值绝对值的和。

原假设是骰子没有问题，因此我们从 values 中随机抽取样本进行模拟。RunModel 使用 Hist 计算和返回频数列表。

计算得到的 $p$ 值为 0.13。也就是说，如果骰子没有问题，我们预期检验统计量达到或超过观察值的概率为 13%。因此，这个直观效应不是统计显著的。

## 9.7 卡方检验

在前一节，我们使用偏差总和作为检测统计量。但是，检测比例时，人们更多使用的是卡方统计量。

$$x^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

其中 $O_i$ 是观察到的频数，$E_i$ 是预期频数。计算卡方统计量的 Python 代码如下：

```
class DiceChiTest(DiceTest):

    def TestStatistic(self, data):
```

```
        observed = data
        n = sum(observed)
        expected = np.ones(6) * n / 6
        test_stat = sum((observed - expected)**2 / expected)
        return test_stat
```

将差值求平方（而不是取绝对值）使大偏差值的权重更大。除以 expected 可以将偏差标准化，虽然在这个例子中预期频数都相同，这种做法没有什么效果。

使用卡方统计量计算的 $p$ 值为 0.04，明显小于使用偏差和的值 0.13。如果我们坚持使用 5% 的阈值，就会认为受检测的效应是统计显著的。但是，将这两个检验放在一起考虑，我认为结果并不明确。我无法排除骰子有问题的可能性，也不能肯定骰子一定有问题。

这个示例说明了一个重要问题：$p$ 值取决于检验统计量的选择和原假设模型，有时这些因素决定了一个效应是否统计显著。

## 9.8　再谈第一胎

我们在本章稍前部分研究了第一胎和其他胎的妊娠时间，认为两组样本均值和标准差的直观差异不是统计显著的。但在 3.3 节，我们看到了妊娠时间分布的几个直观差异，在 35~43 周范围内差异尤为明显。要判断这些差异是否统计显著，我们可以使用基于卡方统计量的检验。

下面一段代码结合了前几个示例的功能。

```
class PregLengthTest(thinkstats2.HypothesisTest):

    def MakeModel(self):
        firsts, others = self.data
        self.n = len(firsts)
        self.pool = np.hstack((firsts, others))

        pmf = thinkstats2.Pmf(self.pool)
        self.values = range(35, 44)
        self.expected_probs = np.array(pmf.Probs(self.values))

    def RunModel(self):
        np.random.shuffle(self.pool)
        data = self.pool[:self.n], self.pool[self.n:]
        return data
```

检验的数据为两组妊娠时间值。我们使用的原假设是两个样本来自同一分布。MakeModel 使用 hstack 将两个样本混合在一起，对分布建模。RunModel 随后将混合样本打乱重排，分为两部分，生成模拟数据。

MakeModel 还定义了检验使用的周数范围 values，以及混合分布中每个值的概率 expected_probs。

下面一段代码计算检验统计量。

```
# class PregLengthTest:

    def TestStatistic(self, data):
        firsts, others = data
        stat = self.ChiSquared(firsts) + self.ChiSquared(others)
        return stat

    def ChiSquared(self, lengths):
        hist = thinkstats2.Hist(lengths)
        observed = np.array(hist.Freqs(self.values))
        expected = self.expected_probs * len(lengths)
        stat = sum((observed - expected)**2 / expected)
        return stat
```

`TestStatistic` 计算第一胎和其他胎的卡方统计量，并求和。

`ChiSquared` 以一个妊娠时间序列为参数，计算直方图及对应 `self.values` 的频数列表 `observed`，并将预期概率 `expected_probs` 乘以样本大小，计算出预期频数列表，返回值为卡方统计量 `stat`。

全国家庭增长调查数据的总卡方统计量为 102，这个数字本身没有什么意义。但在原假设下，1000 次重复产生的最大验证统计量是 32。对比之下，我们认为观察到的卡方统计量在原假设下不太可能出现，因此这个直观效应是统计显著的。

这个示例说明卡方检验存在一个局限：*卡方检验*（chi-squared test）可以证明两个群组之间存在差异，但不能揭示这个差异是什么。

## 9.9 误差

在经典假设检验中，如果 $p$ 值低于某个阈值（常用阈值为 5%），那么我们就认为一个效应是统计显著的。这个过程产生了两个问题。

- 如果一个效应的确是偶然产生的，那么我们将它误判为统计显著的概率是多少？这个概率就是*误报率*（false positive rate）。
- 如果一个效应不是偶然的，那么假设检验失败的概率是多少？这个概率称为*漏报率*（false negative rate）。

相对而言，漏报率比较容易计算。如果阈值为 5%，那么漏报率就是 5%，理由如下。

- 如果效应不为真，那么原假设成立，因此，通过模拟原假设就可以计算出检验统计量的分布。我们将这个分布称为 $CDF_T$。
- 重复运行实验，每次得到一个来自 $CDF_T$ 的测试统计量 $t$。随后，我们计算出 $p$ 值。$p$ 值是 $CDF_T$ 中的随机值大于 `t` 的概率，即为 $1-CDF_T(t)$。

- 当 $CDF_T(t)$ 大于 95%，即 $t$ 大于第 95 百分秩时，$p$ 值小于 5%。而 $CDF_T$ 中随机抽取的值大于第 95 百分秩的概率为多少呢？答案是 5%。

因此，如果你进行一个阈值为 5% 的假设检验，20 次里会有 1 次漏报。

## 9.10 功效

误报率受实际效应大小的影响，而通常我们无法得知实际效应的大小，因此误报率较难计算。一个办法是计算一个假定效应大小的误报率。

举个例子，如果我们假设观测到的组间差异是准确的，那么可以以观测样本为总体模型，使用模拟数据运行假设检验。

```
def FalseNegRate(data, num_runs=100):
    group1, group2 = data
    count = 0

    for i in range(num_runs):
        sample1 = thinkstats2.Resample(group1)
        sample2 = thinkstats2.Resample(group2)

        ht = DiffMeansPermute((sample1, sample2))
        pvalue = ht.PValue(iters=101)
        if pvalue > 0.05:
            count += 1

    return count / num_runs
```

`FalseNegRate` 方法的参数 `data` 是两个序列，每组一个。每次循环从两组中各抽取一个随机样本，运行假设检验，以此模拟一次实验，然后检查检验结果，计算误报次数。

`Resample` 方法以一个序列为参数，使用放回抽样，从中抽取相同长度的样本。

```
def Resample(xs):
    return np.random.choice(xs, len(xs), replace=True)
```

检验妊娠时间的代码如下：

```
    live, firsts, others = first.MakeFrames()
    data = firsts.prglngth.values, others.prglngth.values
    neg_rate = FalseNegRate(data)
```

结果约为 70%。这个结果说明，如果妊娠时间均值的实际差异为 0.78 周，那么我们预期，如果使用这个规模的样本进行实验，结果有 70% 的可能性为误报。

人们经常用另一种方式描述这个结果：如果实际差异为 0.78 周，那么我们预期检验通过的可能性只有 30%。这个“正确通过率”称为检验的*功效*（power），有时也称为“敏感度”

(sensitivity)。这个值反映了一个检验检测出指定大小效应的能力。

在这个示例中，这个检验结果通过的可能性只有 30%（假设实际差异为 0.78 周）。一般说来，我们认为 80% 的功效是可接受的，因此示例中的检验属于“低功效的”(underpowered)。

通常，假设检验失败并不说明两个群组之间不存在差异，而是说，如果差异的确存在的话，这个差异太小，以至于无法在这种规模的样本中检测到。

## 9.11 复现

严格说来，我在本章演示的假设检验过程并非最佳实践。

首先，我进行了多重检验。如果你运行一个假设检验，那么误报的可能性约为 1/20，还在可接受范围内。但是，如果运行 20 个检验，那么在大多数情况下，你至少应该预期得到 1 次误报。

其次，我使用同一个数据集进行探索和检验。如果你对一个大数据集进行探索性研究，发现了一个惊人的效应，然后又检验这个效应是否显著，结果十有八九会是误报。

要弥补多重检验的问题，你可以调整 $p$ 值的阈值（参见 https://en.wikipedia.org/wiki/Holm-Bonferroni_method）。你也可以选择将数据分区，一部分数据用来探索，另一部分用来检验，如此，上面提到的问题都可以得到解决。

在某些领域，这些弥补或解决方法是必须采取的，或者至少是受到鼓励的。但是我们通常也可以通过重现别人发表的结果，间接解决这些问题。通常，人们将报告某个新发现的第一篇论文视为探索性的，使用新数据复现该结果的后续论文则为验证性的。

实际上，我们的确有机会复现本章的结果。本书第一版使用的是全国家庭增长调查的第 6 周期数据，发布于 2002 年。在 2011 年 10 月，疾病控制和预防中心发布了基于 2006~2010 年调查的附加数据。nsfg2.py 中包含读取和清洗这些数据的代码。在这个新数据集中：

- 妊娠时间均值差为 0.16 周，$p< 0.001$，是统计显著的（原始数据中均值差为 0.078 周）；
- 新生儿体重的差值为 0.17 磅，$p< 0.001$（原始数据中差值为 0.12 磅）；
- 新生儿体重与母亲年龄之间的相关性为 0.08，$p< 0.001$（原始数据中相关性为 0.07）；
- 卡方差检验结果 $p< 0.001$，是统计显著的（原始数据中也是如此）。

总之，原始数据中所有统计显著的效应，在新数据集中都得到了复现。此外，妊娠时间差值在原始数据中不显著，在新数据集中这个差值变大，由不显著变为显著。

## 9.12 练习

本章练习的参考答案位于 chap09soln.py 中。

- 练习 9.1

随着样本规模变大，假设检验的功效也会增加，即在效应为真时检验通过的可能性更大。反过来，随着样本规模变小，即便效应为真，检验通过的可能性也会变小。

为了对此进行调查，请使用全国家庭增长调查数据的不同子集运行本章的检验。你可以使用 `thinkstats2.SampleRows`，从 DataFrame 中随机选取一个数据集。

如果样本规模变小，这些检验的 $p$ 值会如何变化？检验能够通过的最小样本规模为多少？

- 练习 9.2

在 9.3 节，我们使用置换对原假设进行了模拟，即用观测值代表整个总体，将总体中的成员随机分配到两个组中。

另一个方法是使用样本估计总体分布，然后从这个分布中抽取随机样本。这一过程称为重抽样（resampling）。重抽样有几种实现方法，最简单的是对观测值进行放回抽样（如 9.10 节所示）。

请编写一个 `DiffMeansResample` 类，继承 `DiffMeansPermute`，重写 `RunModel` 方法，不使用置换，而是实现重抽样。

请使用这个方法，检验妊娠时间和新生儿体重的差值。新模型对结果有何影响？

## 9.13 术语

- 假设检验（hypothesis testing）
  判断一个直观效应是否统计显著的过程。

- 检验统计量（test statistic）
  用于量化效应规模的统计量。

- 原假设（null hypothesis）
  一个系统模型，假设一个直观效应是偶然产生的。

- $p$ 值（p-value）
  一个效应可能偶然产生的概率。

- 统计显著（statistically significant）
  如果一个效应不太可能偶然产生，则是统计显著的。

- 置换检验（permutation test）
  通过重新排列观测数据集计算 $p$ 值。

- 重抽样检验（resampling test）
  通过对观测数据集进行放回抽样，计算 $p$ 值。

- 双侧检验（two-sided test）
  此检验回答的问题是，如果不考虑差异的方向性，实际效应与直观效应规模相同的概率是多少？

- 单侧检验（one-sided test）
  此检验回答的问题是，实际效应与直观效应规模相同并且差异的方向也一致的概率是多少？

- 卡方检验（chi-squared test）
  使用卡方统计量作为检验统计量的检验。

- 误报（false positve）
  当效应为假时，作出效应为真的结论。

- 漏报（false negative）
  当效应非偶然产生时，作出效应是偶然产生的结论。

- 功效（power）
  当原假设为假时，正检验的概率。

# 第10章 线性最小二乘法

本章代码位于 linear.py 中。前言介绍了如何下载和运行本书代码。

## 10.1 最小二乘法拟合

相关系数度量变量关系的强弱和正负，但并不关注关系的斜率。估计斜率有几种方法，最常用的是*线性最小二乘法拟合*（linear least squares fit）。“线性拟合”是用一条线对变量关系进行建模。“最小二乘法”拟合实现线与数据之间的均方差最小。

假设我们要将一个点序列 `ys` 表示成另一个序列 `xs` 的函数。如果 `xs` 和 `ys` 之间存在线性关系，截距为 `inter`，斜率为 `slope`，那么我们就可以预期每个 `y[i]` 值为 `inter + slope * x[i]`。

但是，除非完全相关，否则我们的预测只能是近似的。实际数据到拟合线的竖直偏移或*残差*（residual）为：

```
res = ys - (inter + slope * xs)
```

残差可能由随机因素导致，如测量误差，也可能由未知的非随机因素导致。例如，如果我们试图预测通过身高计算体重的函数，那么未知因素可能有饮食、锻炼及体型等。

如果参数 `inter` 和 `slope` 的估计错误，那么残差就会变大，因此我们很自然地希望选择的参数能使残差最小。

我们可以选择使残差的绝对值最小、平方值最小，或者立方值最小，但是最常见的做法是使残差平方和 `sum(res**2))` 最小。

为什么这样做呢？我们有 3 个很好的理由，还有一个不那么重要的理由：

- 平方值和残差的正负没有关系，这正是我们通常希望的；
- 平方值使较大的残差具有更多的权重，但又不至于使最大的残差作用过大；
- 如果残差不相关，符合均值为 0 且方差为常数（但未知）的正态分布，那么最小二乘法拟合也是 `inter` 和 `slope` 的最大似然估计量（参见 https://en.wikipedia.org/wiki/Linear_regression）；
- 使残差平方最小的 `inter` 和 `slope` 的值有很好的算法。

当考虑计算效率比选择解决问题的最佳方法更重要时，上面列出的最后一个理由就有其存在的道理。但现在我们已经不需要过份受限于计算效率，因此有必要考虑是否应该选择使残差平方最小。

例如，如果你使用 `xs` 预测 `ys` 的值，估计过高可能比估计过低更好（或更差）。在这种情况下，你可能需要对每个残差计算一些成本函数，使总成本 `sum(cost(res))` 最小。不管怎样，最小二乘法拟合计算速度快，容易实现，而且通常结果也不错。

## 10.2 实现

`thinkstats2` 提供了一些简单函数，演示线性最小二乘法的实现。

```
def LeastSquares(xs, ys):
    meanx, varx = MeanVar(xs)
    meany = Mean(ys)

    slope = Cov(xs, ys, meanx, meany) / varx
    inter = meany - slope * meanx

    return inter, slope
```

`LeastSquares` 以序列 `xs` 和 `ys` 为参数，返回估计参数 `inter` 和 `slope`。至于其中的原理，可以参考 http://wikipedia.org/wiki/Numerical_methods_for_linear_least_squares。

`thinkstats2` 还提供 `FitLine` 函数，它以 `inter` 和 `slope` 为参数，返回序列 `xs` 的拟合线。

```
def FitLine(xs, inter, slope):
    fit_xs = np.sort(xs)
    fit_ys = inter + slope * fit_xs
    return fit_xs, fit_ys
```

我们可以使用这些函数，计算由母亲年龄得到新生儿体重函数的最小二乘法拟合。

```
live, firsts, others = first.MakeFrames()
live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
ages = live.agepreg
weights = live.totalwgt_lb
```

```
inter, slope = thinkstats2.LeastSquares(ages, weights)
fit_xs, fit_ys = thinkstats2.FitLine(ages, inter, slope)
```

计算得到的截距和斜率估计值分别为 6.8 磅和每年 0.017 磅。如果我们将截距解释为母亲 0 岁时新生儿的预期体重，这听起来没什么道理，而斜率又非常小，实在很难解释。

我们可以抛开 $x = 0$ 时的截距，转而讨论在 $x$ 均值处的截距。在我们的数据中，母亲的年龄均值为 25 岁，那么 25 岁母亲产下的婴儿的平均体重为 7.3 磅。斜率为每年 0.27 盎司，或每 10 年 0.17 磅。

图 10-1 展示了新生儿体重和母亲年龄的散点图以及拟合线。查看这种图表，我们可以估计变量关系是否为线性，拟合线是否很好地反映了变量关系。这通常是种不错的做法。

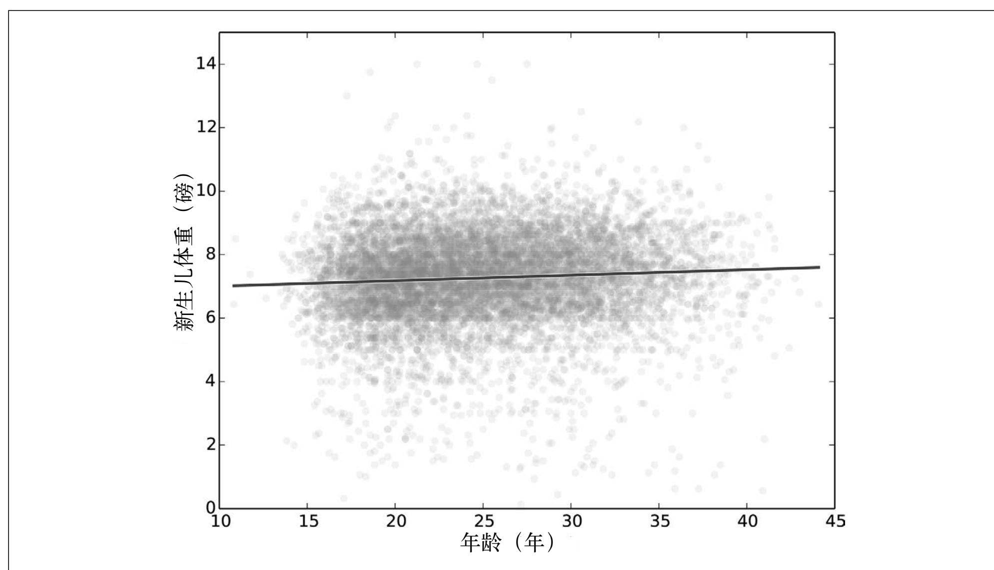



图 10-1：新生儿体重与母亲年龄的散点图及线性拟合

# 10.3　残差

还有一种有用的检验方法，即绘制残差。thinkstats2 提供一个计算残差的函数。

```
def Residuals(xs, ys, inter, slope):
    xs = np.asarray(xs)
    ys = np.asarray(ys)
    res = ys - (inter + slope * xs)
    return res
```

Residuals 的参数是序列 xs 和 ys，以及估计参数 inter 和 slope。Residuals 返回实际值和

拟合线之间的差值。

为了对残差进行可视化展示，我把调查参与者按年龄分组，用 7.2 节中的做法计算每组的百分位秩。图 10-2 展示了每个年龄组残差的第 25、第 50 和第 75 百分位秩。正如我们预期的，中位数值接近 0，四分位距约为 2 磅。因此，如果我们已知母亲的年龄，就可以有 50% 的概率将新生儿的体重估计精确到磅。

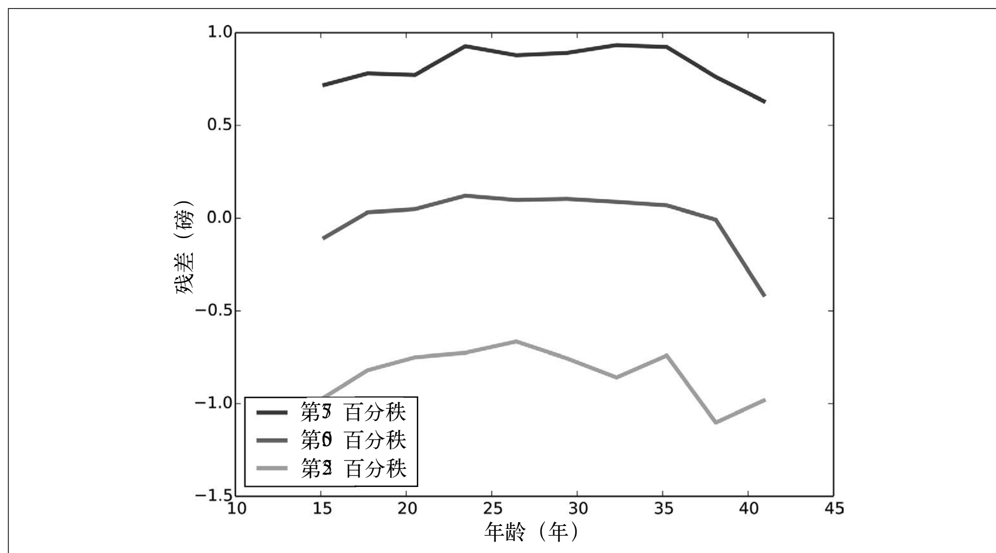



图 10-2：线性拟合的残差

理想情况下，这些线应该是平的，说明残差是随机的；这些线还应该是平行的，说明所有年龄组的残差方差都很小。实际上，图中这些线接近平行，这一点很不错。但是这些线都弯弯曲曲的，说明关系是非线性的。尽管如此，线性拟合仍然是可能满足某些需求的不错的简单模型。

# 10.4 估计

参数 `slope` 和 `inter` 是基于样本估计得到的，因而也和其他估计值一样，会受到抽样偏倚、测量误差和抽样误差的影响。我们在第 8 章讨论过，抽样偏倚是由不具代表性的样本导致的，测量误差是在数据收集和记录过程中产生的，测量样本而非测量总体则会产生抽样误差。

为了估计抽样误差，我们会问："如果再次运行这个实验，我们预期估计值会有多大变化？"要回答这个问题，我们可以运行模拟实验，计算估计值的抽样分布。

我模拟实验的方法是对数据进行重抽样，也就是说，将观测到的妊娠数据作为整个总体，

从这个观测样本中进行放回抽样。

```
def SamplingDistributions(live, iters=101):
    t = []
    for _ in range(iters):
        sample = thinkstats2.ResampleRows(live)
        ages = sample.agepreg
        weights = sample.totalwgt_lb
        estimates = thinkstats2.LeastSquares(ages, weights)
        t.append(estimates)

    inters, slopes = zip(*t)
    return inters, slopes
```

`SamplingDistributions` 的参数之一为 DataFrame，其中每个成功生产数据为一行，另一个参数是待模拟的实验次数 `iters`。`SamplingDistributions` 使用 `ResampleRows` 方法对观测到的妊娠数据进行重抽样。我们之前使用过 `SampleRows` 方法，从一个 DataFrame 中抽取随机的数据行。`thinkstats2` 也提供 `ResampleRows` 方法，这个方法返回的样本与原样本大小相同。

```
def ResampleRows(df):
    return SampleRows(df, len(df), replace=True)
```

完成重抽样后，我们使用模拟样本来估计参数。结果为两个序列：截距估计值和斜率估计值。

最后，我们将这个抽样分布的标准误差和置信区间打印出来。

```
def Summarize(estimates, actual=None):
    mean = thinkstats2.Mean(estimates)
    stderr = thinkstats2.Std(estimates, mu=actual)
    cdf = thinkstats2.Cdf(estimates)
    ci = cdf.ConfidenceInterval(90)
    print('mean, SE, CI', mean, stderr, ci)
```

`Summarize` 的参数是估计值和实际值序列，打印估计值的均值、标准误差和 90% 置信区间。

我们计算得到截距估计值的均值为 6.83，标准误差为 0.07，90% 置信区间为 (6.71, 6.94)。斜率估计值的结果可以写为：0.0174、SE 0.0028 和 CI (0.0126, 0.0220)。斜率估计值 CI 的最大值几乎是最小值的两倍，应视为粗略估计。

为了可视化地展示这个估计的抽样误差，我们可以绘制出所有的拟合线。或者，简洁起见，我们可以绘制每个年龄的 90% 置信区间。代码如下：

```
def PlotConfidenceIntervals(xs, inters, slopes,
                            percent=90, **options):
    fys_seq = []
    for inter, slope in zip(inters, slopes):
        fxs, fys = thinkstats2.FitLine(xs, inter, slope)
```

```
        fys_seq.append(fys)

    p = (100 - percent) / 2
    percents = p, 100 - p
    low, high = thinkstats2.PercentileRows(fys_seq, percents)
    thinkplot.FillBetween(fxs, low, high, **options)
```

代码中的 xs 是母亲年龄序列，inters 和 slopes 是 SamplingDistributions 生成的参数估计值，percent 指定了需要绘制的置信区间。

PlotConfidenceIntervals 为每一对 inter 和 slope 值生成一条拟合线，并将结果存储在 fys_seq 序列中，然后使用 PercentileRows 为每个 x 值选择 y 的高低百分位秩。对于 90% 的置信区间，PlotConfidenceIntervals 选择第 5 和第 95 百分位秩。FillBetween 方法在两条拟合线之间绘制一个多边形进行填充。

对于新生儿体重与母亲年龄关系的拟合曲线，图 10-3 展示了其 50% 和 90% 置信区间。图中区域的竖直宽度代表抽样误差的影响。抽样误差对均值附近的值影响较小，对极端值影响较大。

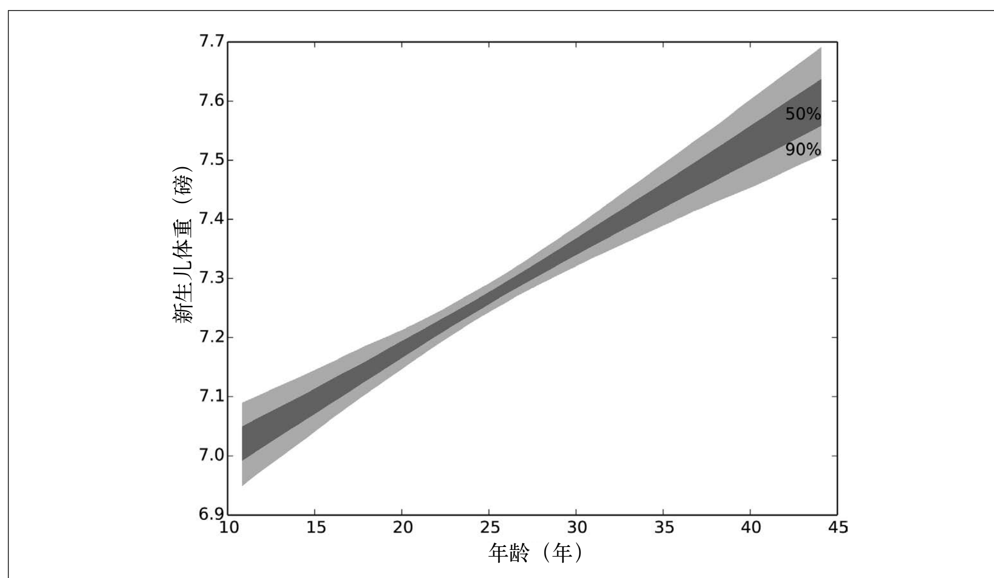



图 10-3：50% 和 90% 置信区间，展示了 inter 和 slope 抽样误差导致的拟合线变化

# 10.5　拟合优度

要度量一个线性模型的质量优劣，或拟合优度（goodness of fit），我们可以采取好几种方法，最简单的方法之一是度量残差的标准差。

如果使用线性模型进行预测，那么 `Std(res)` 就是预测值的均方根误差，即均方误差的平方根。例如，如果使用母亲的年龄来估计新生儿的体重，那么估计值的均方根误差就是 1.40 磅。

如果在不知道母亲年龄的情况下估计新生儿体重，那么估计值的均方根误差为 `Std(ys)`，值为 1.41 磅。因此，在我们使用的这个示例中，已知母亲年龄并不能显著提高预测的准确性。

度量拟合优度的另一种方法是计算*决定系数*（coefficient of determination），通常写作 $R^2$，读作“$R$ 平方”。

```
def CoefDetermination(ys, res):
    return 1 - Var(res) / Var(ys)
```

其中 `Var(res)` 是使用模型进行猜测的均方误差，`Var(ys)` 是不使用模型时的均方误差。因此，这两个值的比率是使用模型仍存在的均方误差比例，$R^2$ 是该模型消除的均方差比例。

对于新生儿体重和母亲年龄，$R^2$ 为 0.0047，说明使用母亲年龄大约能预测新生儿体重变化中 1% 的一半。

决定系数和 Pearson 相关系数之间有一个简单的关系：$R^2=p^2$。例如，如果 $p$ = 0.8 或 −0.8，那么 $R^2$= 0.64。

虽然人们经常使用 $p$ 和 $R^2$ 量化变量关系的强弱，但是用它们解释预测能力却很困难。在我看来，`Std(res)` 是预测能力的最佳体现，尤其是和 `Std(ys)` 一起使用时。

例如，当人们谈到 SAT（美国大学录取使用的标准化测试）的有效性时，经常会讨论 SAT 成绩与其他智力测验之间的相关性。

一项研究表明，SAT 总成绩与 IQ 值之间的 Person 相关性为 $p$ = 0.72，二者看似具有很强的相关性。但是 $R^2=p^2$= 0.52，因此 SAT 成绩只能影响 IQ 值变化的 52%。

将 IQ 值正态化，得到 `Std(ys) = 15`，因此：

```
>>> var_ys = 15**2
>>> rho = 0.72
>>> r2 = rho**2
>>> var_res = (1 - r2) * var_ys
>>> std_res = math.sqrt(var_res)
10.4096
```

因此，如果我们使用 SAT 成绩预测 IQ 值，可以将均方根误差从 15 降低到 10.4。值为 0.72 的相关性只能将均方根误差降低约 31%。

如果你看到一个看似很强的相关性，请记住 $R^2$ 更能反映均方误差的降低程度，而均方根误差的降低程度可以更好地说明预测能力。

## 10.6　检验线性模型

母亲年龄对新生儿体重的影响很小，几乎没有预测能力。那么，这种明显的关系可能是偶然产生的吗？我们可以使用几种方法，检验线性拟合的结果。

一种方法是检验均方误差的显著降低是否偶然。在这个检验中，检验统计量为 $R^2$，原假设为变量之间不存在关系。我们可以采用 9.5 节中检验母亲年龄与新生儿体重相关性时的做法，通过置换对原假设进行模拟。实际上，由于 $R^2 = p^2$，$R^2$ 的单侧检验等效于 $p$ 的双侧检验。我们已经进行过这项检验，得到 $p < 0.001$，因此认为母亲年龄和新生儿体重之间关系的直观效应是统计显著的。

另一种方法是检验斜率是否偶然。在这个检验中，原假设是斜率实际为 0。因此，我们可以使用均值附近的随机变化建立新生儿体重模型。这个模型的假设检验实现如下：

```
class SlopeTest(thinkstats2.HypothesisTest):

    def TestStatistic(self, data):
        ages, weights = data
        _, slope = thinkstats2.LeastSquares(ages, weights)
        return slope

    def MakeModel(self):
        _, weights = self.data
        self.ybar = weights.mean()
        self.res = weights - self.ybar

    def RunModel(self):
        ages, _ = self.data
        weights = self.ybar + np.random.permutation(self.res)
        return ages, weights
```

代码使用的数据为年龄和体重序列。检验统计量为 `LeastSquares` 估计的斜率。代码使用所有新生儿体重的均值和到均值的偏差表示原假设模型。代码对偏差进行置换，然后叠加在均值上，从而生成模拟数据。

运行这个假设检验的代码如下：

```
    live, firsts, others = first.MakeFrames()
    live = live.dropna(subset=['agepreg', 'totalwgt_lb'])
    ht = SlopeTest((live.agepreg, live.totalwgt_lb))
    pvalue = ht.PValue()
```

得到的 $p$ 值小于 0.001。因此，虽然估计斜率很小，但不太可能是偶然产生的。

通过模拟原假设来估计 $p$ 值是非常正确的，但还有更简单的方法。请记住，我们在 10.4 节已经计算过斜率的抽样分布。计算时，我们假设观测斜率是正确的，使用重抽样进行模拟实验。

图 10-4 展示了 10.4 节中得到的斜率抽样分布，以及原假设下生成的斜率分布。抽样分布的中间值约为估计斜率，即 0.017 磅 / 年。原假设下的斜率中间值约为 0。除此之外，这两个分布是一样的，而且还是对称的（原因将在 14.4 节中进行讨论）。

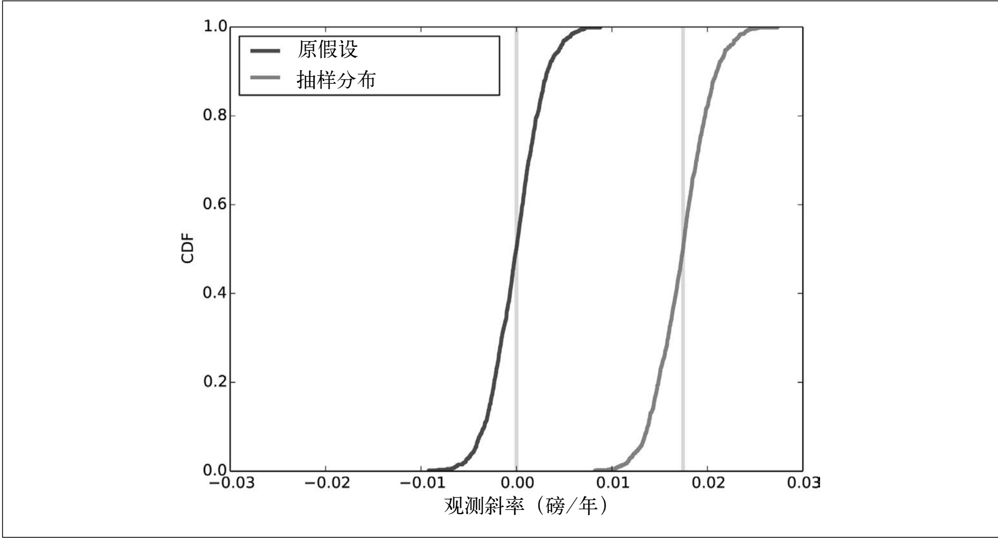



图 10-4：估计斜率的抽样分布及在原假设下生成的斜率分布，竖线分别位于 0 和观测斜率（即 0.017 磅 / 年）处

因此，我们可以用两种方法估计 $p$ 值。

- 计算原假设下斜率大于观测斜率的概率。
- 计算抽样分布中斜率小于 0 的概率。（如果估计斜率为负数，那么应该计算抽样分布中斜率大于 0 的概率。）

我们通常会计算参数的抽样分布，因此第二种方法较为容易。而且第二种方法计算的值较为准确，除非样本规模很小而且残差分布偏斜。即便如此，第二种方法计算的值通常也很不错，因为 $p$ 值并不需要特别准确。

下面的代码使用抽样分布估计斜率的 $p$ 值。

```
inters, slopes = SamplingDistributions(live, iters=1001)
slope_cdf = thinkstats2.Cdf(slopes)
pvalue = slope_cdf[0]
```

结果再一次得到 $p < 0.001$。

## 10.7　加权重抽样

到目前为止，我们一直把全国家庭增长调查数据当作代表性样本，然而 1.2 节中提到过，

这部分数据并不是代表性样本。这个调查特意对一些群组进行过度抽样，以提高获得统计显著结果的机会，即增加了涉及这些群组的检验功效。

这种调查设计有很多用途，但也意味着如果不考虑具体的抽样过程，我们就无法使用这个样本估计总体。

对于每个调查参与者，全国家庭增长调查数据都包含一个变量 finalwgt，这是该参与者代表的全国人口数量。这个值称为*抽样权重*（sampling weight），或直接称为“权重”。

举个例子，假设某个国家有 3 亿人，而我们调查其中的 10 万人，那么每个调查参与者都代表 3000 人，如果对一个群组抽取了两倍的调查者，那么这个过度抽样群组中的每个人就具有较低的权重，约为 1500。

要矫正过度抽样，我们可以重抽样，即按照与抽样权重成比例的概率从调查结果中抽取样本，然后对需要估计的数值生成抽样分布、标准误差和置信区间。我们以新生儿体重为例，采取使用和不使用抽样权重两种方法，估计新生儿体重均值，并将结果进行对比。

10.4 节用到了 ResampleRows，这个方法从一个 DataFrame 中选取数据行，每行被抽中的概率相同。现在我们需要使用与抽样权重成比例的概率再次进行抽样。ResampleRowsWeighted 方法以一个 DataFrame 为参数，按照 finalwgt 中的权重对数据行进行重抽样，返回的 DataFrame 中包含重抽样后的数据行。

```
def ResampleRowsWeighted(df):
    weights = df.finalwgt
    pmf = thinkstats2.Pmf(weights.iteritems())
    cdf = pmf.MakeCdf()
    indices = cdf.Sample(len(weights))
    sample = df.loc[indices]
    return sample
```

pmf 将每行的索引映射到正态化权重。代码将概率函数转换为累积分布函数，从而缩短抽样过程的时间。indices 是列索引序列；sample 是包含选中数据行的 DataFrame。由于代码使用放回抽样，因此同一行可能出现多次。

现在，我们可以将使用权重和不使用权重的重抽样效果进行对比。如果不使用权重，生成抽样分布的代码如下：

```
estimates = [ResampleRows(live).totalwgt_lb.mean()
             for _ in range(iters)]
```

使用权重生成抽样分布的代码为：

```
estimates = [ResampleRowsWeighted(live).totalwgt_lb.mean()
             for _ in range(iters)]
```

下表列出了抽样结果的统计量。

| | 新生儿体重均值（磅） | 标准误差 | 90%置信区间 |
|---|---|---|---|
| 不使用权重 | 7.27 | 0.014 | (7.24, 7.29) |
| 使用权重 | 7.35 | 0.014 | (7.32, 7.37) |

本例中，权重的效果很小，但却不容忽视。两种抽样方式得到的估计均值区别约为 0.08 磅，或 1.3 盎司。这个区别显著大于这个估计值的标准误差 0.014 磅，说明这种区别不是偶然产生的。

## 10.8 练习

本章练习的参考答案位于 chap10soln.ipynb 中。

练习 10.1

请使用 BRFSS 数据，计算体重的对数值与身高的线性最小二乘法拟合。在这个模型中，一个变量经过对数计算转换后，如何才能最佳展示估计参数？如果需要估计某人体重，已知身高会带来多大帮助？

和全国家庭增长调查一样，BRFSS 也对某些群组进行了过度抽样，并对每个调查参与者提供抽样权重。在 BRFSS 数据中，抽样权重的变量名为 `totalwt`。请使用带权重和不带权重的重抽样估计 BRFSS 调查参与者的身高均值、该均值的标准误差以及 90% 置信区间。使用权重对这个估计值有何影响？

## 10.9 术语

- 线性拟合（linear fit）
  对变量关系进行建模的线。
- 最小二乘法拟合（least squares fit）
  使残差平方和最小的数据集模型。
- 残差（residual）
  实际值与模型的偏差。
- 拟合优度（goodness of fit）
  度量模型与数据相符合的程度。
- 决定系数（coefficient of determination）
  量化拟合优度的统计量。
- 抽样权重（sampling weight）
  与样本中一个观测相关的值，说明该观测代表总体的哪一部分。

# 第11章 回归

前一章介绍的线性最小二乘法拟合属于*回归*（regression）的一种。回归的范围更广，涵盖将任何模型拟合到任何数据的问题。“回归”这个词的使用属于历史遗留问题，本章“回归”一词的含义和其原始含义并不完全一致，只是间接相关。

回归分析的目的是描述两组变量之间的关系，一组称为*因变量*（dependent variable），另一组称为*解释变量*（explanatory variable）。

在前一章，我们以母亲年龄为解释变量，预测因变量新生儿的体重。如果回归分析中只有 1 个因变量和 1 个解释变量，就属于*简单回归*（simple regression）。本章将讨论*多重回归*（multiple regression），涉及多个解释变量。有多个因变量的回归分析称为*多元回归*（multivariate regression）。

如果因变量和解释变量之间的关系是线性的，就属于*线性回归*（linear regression）。例如，如果因变量为 $y$，解释变量为 $x_1$ 和 $x_2$，那么我们可以写出如下的线性回归模型：

$$y=\beta_0+\beta_1x_1+\beta_2x_2+\varepsilon$$

其中 $\beta_0$ 为截距，$\beta_1$ 是与 $x_1$ 相关的参数，$\beta_2$ 是与 $x_2$ 相关的参数，$\varepsilon$ 是随机变化或其他未知因素导致的残差。

给定一列 $y$ 值，以及 $x_1$ 和 $x_2$ 的值，我们可以算出使得 $\varepsilon^2$ 最小的参数值 $\beta_0$、$\beta_1$ 和 $\beta_2$。这个过程称为*普通最小二乘法*（ordinary least square）。普通最小二乘法的计算方法与 `thinkstats2.LeastSquare` 类似，但更为通用，可以处理多个解释变量。细节请参考 https://en.wikipedia.org/wiki/Ordinary_least_squares。

本章代码位于 regression.py 中。前言介绍了如何下载和使用本书代码。

## 11.1 StatsModels

前一章介绍了 thinkstats2.LeastSquares，这个方法实现了简单线性回归，且代码非常易读。对于多重回归，我们将使用 StatsModels，这个 Python 包提供了几种回归形式和其他分析方法。Anaconda 自带 StatsModels 包，如果你使用的不是 Anaconda，那么就需要单独安装。

下面的代码使用 StatsModels 运行前一章的模型：

```
import statsmodels.formula.api as smf

live, firsts, others = first.MakeFrames()
formula = 'totalwgt_lb ~ agepreg'
model = smf.ols(formula, data=live)
results = model.fit()
```

statsmodels 提供 2 个接口（API）。“公式”API 使用字符串标识因变量和解释变量，所用语法称为 patsy。这个示例使用操作符 ~ 分隔不同的变量：左边是因变量，右边是解释变量。

smf.ols 以公式字符串和 DataFrame live 为参数，返回表示模型的 OLS 对象。ols 代表“普通最小二乘法”。

fit 方法将模型拟合到数据，返回一个 RegressionResults 对象，其中包含拟合结果。

拟合结果也可以通过属性获得。params 是一个 Series 对象，将变量名映射到对应的参数，因此，我们可以用如下代码获得截距和斜率：

```
inter = results.params['Intercept']
slope = results.params['agepreg']
```

估计参数值为 6.83 和 0.017 5，与使用 LeastSquares 得到的结果一样。

pvalues 是一个 Series 对象，将变量名映射到相关的 $p$ 值，因此我们可以检查斜率估计值是否是统计显著的：

```
slope_pvalue = results.pvalues['agepreg']
```

与 agereg 相关的 $p$ 值为 5.7e−11，小于 0.001，和预期一样。

results.rsquared 为 $R^2$ 值，结果为 0.0047。results 还提供 f_pvalue，这是与整个模型相关的 $p$ 值，与检验 $R^2$ 是否统计显著类似。

另外，results 还提供残差序列 resid，以及对应于 agepreg 的拟合值序列 fittedvalues。

results 对象提供 summary() 方法，将结果表示为可读格式。

```
print(results.summary())
```

但是这个打印结果会输出许多无关信息（至少现阶段是无关的），因此可以使用更简单的函数 SummarizeResults。这个模型的结果显示如下：

```
Intercept       6.83    (0)
agepreg         0.0175  (5.72e-11)
R^2 0.004738
Std(ys) 1.408
Std(res) 1.405
```

Std(ys) 是因变量的标准差，即不利用任何解释变量时猜测新生儿体重得到的均方根误差。Std(res) 是残差的标准差，即使用母亲年龄对新生儿体重进行猜测时得到的均方根误差。正如我们之前看到的，已知母亲年龄并不能显著提高新生儿体重预测的准确性。

# 11.2　多重回归

在 4.5 节我们看到，第一胎比其他胎的新生儿体重轻，而且这种效应是统计显著的。但是并没有明显机制导致第一胎的体重较轻，所以这个结果有些奇怪。因此，人们可能怀疑这种关系是*虚假的*（spurious）。

实际上，这种效应存在一种可能的解释，即新生儿体重受母亲年龄影响，而生产第一胎时母亲的年龄较小。

我们可以通过一些计算来检验这种解释是否具有说服力，然后使用多重回归进行更细致的调查。首先，让我们看看第一胎与其他胎的新生儿体重的差值是多少：

```
diff_weight = firsts.totalwgt_lb.mean() - others.totalwgt_lb.mean()
```

第一胎比其他胎的新生儿的体重轻 0.125 磅，或 2 盎司。让我们再看看母亲年龄的差值：

```
diff_age = firsts.agepreg.mean() - others.agepreg.mean()
```

产下第一胎的母亲比其他胎新生儿的母亲年龄小 3.59 岁。再次运行线性模型，得到以母亲年龄为变量的新生儿体重函数：

```
results = smf.ols('totalwgt_lb ~ agepreg', data=live).fit()
slope = results.params['agepreg']
```

得到的斜率为 0.175 磅 / 年。将这个斜率乘以母亲年龄差，可以得到由母亲年龄导致的新生儿体重差的预期值：

```
slope * diff_age
```

结果为 0.063，约为观测差值的一半。因此，我们暂时认为，观测到的新生儿体重差值可以部分解释为是由母亲年龄差导致的。

使用多重回归，我们可以更加系统地探索这些关系。

```
live['isfirst'] = live.birthord == 1
formula = 'totalwgt_lb ~ isfirst'
results = smf.ols(formula, data=live).fit()
```

第一行代码创建了一个名为 `isfirst` 的新列，第一胎记录的 `isfirst` 值为真，其他胎记录的 `isfirst` 值为假。随后的代码以 `isfirst` 为解释变量，拟合一个模型。

这段代码生成的结果如下：

```
Intercept         7.33   (0)
isfirst[T.True]  -0.125  (2.55e-05)
R^2 0.00196
```

`isfirst` 是布尔型，因此 `ols` 将其当作*分类变量*（categorical variable），即这些值分为不同类别，如真或假，而不应看作数字。估计参数是 `isfirst` 为真对新生儿体重的影响，因此结果 −0.125 磅代表第一胎和其他胎的新生儿体重的差值。

最终得到的斜率和截距都是统计显著的，说明它们不太可能偶然产生。但这个模型的 $R^2$ 值很小，说明 `isfirst` 对新生儿体重变化的影响不大。

以 `apepreg` 为解释变量得到的结果为：

```
Intercept       6.83    (0)
agepreg         0.0175  (5.72e-11)
R^2 0.004738
```

同样，这些参数是统计显著的，但 $R^2$ 值很小。

这些模型证实了我们之前的结论。但是，现在我们可以用单个模型同时包含这两个变量。使用公式 `totalwgt_lb ~ isfirst + agepreg` 得到的结果为：

```
Intercept        6.91    (0)
isfirst[T.True] -0.0698  (0.0253)
agepreg          0.0154  (3.93e-08)
R^2 0.005289
```

在这个组合模型中，`isfirst` 的参数更小，约是单变量模型中的一半，说明 `isfirst` 的这部分直观效应实际是由 `agepreg` 产生的。`isfirst` 的 $p$ 值约为 2.5%，处在统计显著的边缘。

这个模型的 $R^2$ 值略高，说明这两个变量对新生儿体重的共同影响要大于其中任何一个的单独影响（但大得不多）。

## 11.3　非线性关系

由于 agepreg 对新生儿体重的影响可能不是线性的，所以可以考虑添加一个变量，从而更好地反映这种关系。我们可以添加一列 agepreg2，值为母亲年龄的平方：

```
live['agepreg2'] = live.agepreg**2
formula = 'totalwgt_lb ~ isfirst + agepreg + agepreg2'
```

通过估计 agepreg 和 agepreg2 的参数，可以有效地拟合一个抛物线：

```
Intercept        5.69     (1.38e-86)
isfirst[T.True] -0.0504   (0.109)
agepreg          0.112    (3.23e-07)
agepreg2        -0.00185  (8.8e-06)
R^2 0.007462
```

agepreg2 的参数是负值，因此这个抛物线是向下弯曲的，与图 10-2 中的曲线形状相符。

agepreg 的二次模型对新生儿体重变化进行了更多的解释。在这个模型中，isfirst 的参数较小，而且不再是统计显著的。

使用 agepreg2 这样的计算变量，是对数据进行多项式或其他函数拟合的常用方法。虽然有些解释变量是其他变量的非线性函数，但因变量是解释变量的线性函数，因此我们仍然认为这个过程属于线性回归。

这些回归结果如下表所示：

| | isfirst | agepreg | agepreg2 | $R^2$ |
|---|---|---|---|---|
| 模型 1 | −0.125* | — | — | 0.002 |
| 模型 2 | — | 0.0175* | — | 0.0047 |
| 模型 3 | −0.0698(0.025) | 0.0154* | — | 0.0053 |
| 模型 4 | −0.0504(0.11) | 0.112* | −0.00185* | 0.0075 |

表中的列为解释变量和决定系数 $R^2$。每一格都是一个估计参数，其后括号中是 $p$ 值或星号，星号表示 $p$ 值小于 0.001。

我们认为，新生儿体重的直观差异可以（至少部分）由母亲年龄的差异进行解释。如果模型中包括母亲年龄，那么 isfirst 的效应会变小，剩余的效应可能是偶然产生的。

在这个示例中，母亲年龄是*控制变量*（control variable）。模型中包含的 agepreg “控制”了第一次生产和其他情况母亲年龄的差异，从而将 isfirst 的效应（如果存在的话）隔离起来。

## 11.4 数据挖掘

到目前为止，我们只将回归模型用于解释。例如，在前一节中，我们发现新生儿体重的直观差异是由母亲年龄的差异导致的。但这些模型的 $R^2$ 值很小，说明其预测能力很弱。这一节我们将尝试一些更好的方法。

假设一位同事有孕待产，办公室设了一个赌局，预测新生儿的体重（如果不了解彩池设赌，请参考 https://en.wikipedia.org/wiki/Betting_pool）。

如果你非常想赢得这个赌局，那么该如何提高自己的胜算呢？在全国家庭增加调查数据集中，每个妊娠记录有 244 个变量，每位参与者还有其他 3087 个变量。也许这些变量中的某些具有预测能力。要找到哪些变量用处最大，为什么不把这些变量都试试呢？

检验妊娠数据表中的变量很容易，但是要使用参与者表中的变量必须将每个妊娠记录与一位调查参与者进行匹配。理论上，我们可以遍历妊娠数据表中的行，使用 `caseid` 找到对应的参与者，然后把参与者数据表中的值复制到妊娠数据表中，但是这种做法会很慢。

更好的方法是使用*连接*（join）操作，SQL 和其他关系数据库语言都定义了连接操作（参见 https://en.wikipedia.org/wiki/Join_(SQL)）。DataFrame 实现了 `join` 方法，连接操作可以如下实现：

```
live = live[live.prglngth>30]
resp = chap01soln.ReadFemResp()
resp.index = resp.caseid
join = live.join(resp, on='caseid', rsuffix='_r')
```

假设办公室赌局设在孩子出生前的几周，因此第一行代码选择妊娠时间超过 30 周的记录。

第二行代码读取调查参与者文件，结果为使用整数索引的 DataFrame。为了高效查找参与者，将 `resp.index` 替换为 `resp.caseid`。

`live` 对象调用 `join` 方法，参数为 `resp` 对象，`live` 为“左”表，`resp` 为“右”表。关键字参数 `on` 说明了两张表中进行匹配的变量。

在这个示例中，有些列名在两张表中都存在，因此 `join` 方法必须提供 `rsuffix` 参数。`rsuffix` 是一个字符串，附加在右表中重复的列名后。例如，这两张表都有一列名为 `race`，是调查参与者的种族编码。`join` 结果包含列 `race` 和 `race_r`。

pandas 的连接操作实现速度非常快。在一台普通台式机上进行全国家庭增长调查表操作，不到 1 秒就可以完成。现在，我们可以开始检验变量。

```
t = []
for name in join.columns:
    try:
```

```
            if join[name].var() < 1e-7:
                continue

            formula = 'totalwgt_lb ~ agepreg + ' + name
            model = smf.ols(formula, data=join)
            if model.nobs < len(join)/2:
                continue

            results = model.fit()
        except (ValueError, TypeError):
            continue

        t.append((results.rsquared, name))
```

对每个变量，构建一个模型，计算其 $R^2$，然后把结果附加到一个列表中。我们已经知道 agepreg 具有一定的预测能力，因此所有的模型都包含 agepreg。

代码检测发现每个解释变量的值都发生了变化，否则回归结果将是不可靠的，此外还对每个模型的观测数量进行了检测。包含大量 nans 的变量不适合用于预测。

我们对大部分的变量都没有进行任何清洗。有些变量的编码方式不太适合线性回归，因此我们可能会错过一些需要清洗的有用变量，但也许能找到一些不错的预测变量。

# 11.5　预测

下一步是对结果进行排序，选择 $R^2$ 值最高的变量。

```
t.sort(reverse=True)
for mse, name in t[:30]:
    print(name, mse)
```

列表中的第一个变量是 totalwgt_lb，第二个是 birthwgt_lb。显然，新生儿体重不能用来预测新生儿体重。

同样，prglngth 具有一些预测能力，但是在这个办公室赌局里，妊娠时间（以及相关变量）还属于未知。

第一个有用的预测变量是 babysex，代表新生儿的性别。在全国家庭增长调查数据集中，男孩大概比女孩重 0.3 磅。因此，如果我们知道新生儿的性别，就可以用于预测体重。

下一个变量是 race，说明调查参与者是白人、黑人或其他种族。种族作为一个解释变量可能有些问题。在全国家庭增长调查这样的数据集中，种族与很多其他变量相关，如收入和其他社会经济因素。在回归模型中，种族是一个*代理变量*（proxy variable），因此与种族相关的直观效应经常是（至少部分）由其他因素导致的。

下一个变量是 nbrnaliv，即是否多胞胎。双胞胎和三胞胎会比其他新生儿的体型小。因

此，如果知道这位假想同事是否怀着双胞胎，对预测结果会有帮助。

再下一个变量是 paydu，即调查参与者是否拥有自己的住宅。paydu 是具有预测能力的几个收入相关变量之一。在全国家庭增长调查这样的数据集中，收入和财富几乎与所有效应都相关。在这个示例中，收入与饮食、健康、保健等其他因素相关，这些因素都可能影响新生儿的体重。

这个列表中其他一些变量的信息要到生产后才会知道，例如婴儿母乳喂养周数 bfeedwks。这些变量无法用于预测，但是你也许会猜想为什么 bfeedwks 会与新生儿体重相关。

有时人们会先形成一个理论，然后用数据检验这个理论，有时则直接用数据搜寻可能的理论。本节使用的就是第二种方法，这种方法称为*数据挖掘*（data mining）。数据挖掘的好处是可以发现不曾预期的模式，但问题是发现的很多模式是随机或虚假的。

找到有潜力的解释变量后，我测试了一些模型，并选定了其中一个。

```
formula = ('totalwgt_lb ~ agepreg + C(race) + babysex==1 + '
           'nbrnaliv>1 + paydu==1 + totincr')
results = smf.ols(formula, data=join).fit()
```

这个公式使用了一些新语法。C(race) 告诉公式解析器（Patsy）将 race 作为分类变量，虽然这个变量编码为数值。

babysex 将男性编码为 1，女性编码为 2。表达式 babysex==1 将 babysex 值转换为布尔值，男性为 True，女性为 False。

同样，多胞胎的 nbrnaliv>1 结果为 True，拥有自己住宅的调查参与者 paydu==1 结果为 True。

totincr 的编码值为 1~14，值每增加 1 代表年收入增加约 5000 美元。因此，我们可以把这些值作为数值处理，用 5000 美元为显示单位。

这个模型的结果如下：

```
Intercept               6.63    (0)
C(race)[T.2]            0.357   (5.43e-29)
C(race)[T.3]            0.266   (2.33e-07)
babysex == 1[T.True]    0.295   (5.39e-29)
nbrnaliv > 1[T.True]   -1.38    (5.1e-37)
paydu == 1[T.True]      0.12    (0.000114)
agepreg                 0.00741 (0.0035)
totincr                 0.0122  (0.00188)
```

模型使用了收入控制变量，但变量 race 的估计参数仍然比预期值大。race 值为 1 代表黑人，2 代表白人，3 代表其他种族。黑人母亲产下的孩子比其他种族的孩子体重轻 0.27 到 0.36 磅。

模型结果显示，和之前讨论的一样，男孩比女孩重大概 0.3 磅，多胞胎比其他新生儿轻 1.4 磅。

即便使用了收入控制变量，模型仍然显示，拥有自己住宅的母亲产下的孩子比其他新生儿重 0.12 磅。母亲年龄的参数比 11.2 节中的结果小，说明其他一些变量与年龄相关，这些变量可能包括 `paydu` 和 `totincr`。

所有这些变量都是统计显著的，有些 $p$ 值很低，但 $R^2$ 值只有 0.06，仍然很小。不使用这个模型时，均方根误差值为 1.27 磅；使用这个模型时，均方根误差降到了 1.23 磅。因此，你赢得这个赌局的机会并没有显著增加，抱歉！

## 11.6 Logistic回归

在之前的示例中，一些解释变量是数值型的，一些是分类型（包括布尔型）的。但因变量都是数值型的。

线性回归可以通用化，从而处理其他种类的因变量。如果因变量是布尔型，那么通用化模型称为 Logistic *回归*（logistic regression）。如果因变量是整数，那么模型称为 Poisson *回归*（Poisson regression）。

举个 Logistic 回归的例子，我们可以考虑办公室赌局的一种变型。假设一位朋友有孕，你想预测孩子的性别。此时可以使用全国家庭增长调查数据寻找影响“性别比例”的因素，而性别比率正是新生儿为男孩的概率。

如果将因变量用数值进行编码（例如，0 代表女孩，1 代表男孩），那么你可以使用普通最小二乘法，但这里存在一些问题。线性模型的表示形式如下：

$$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\varepsilon$$

其中 $y$ 为因变量，$x_1$ 和 $x_2$ 为解释变量。我们之后需要找到使残差最小的参数。

使用普通最小二乘法有一个问题，即所产生的预测难以解释。如果给定预测参数，以及 $x_1$ 和 $x_2$ 的值，那么这个模型可能预测得到 $y = 0.5$，而 $y$ 有意义的取值仅为 0 或 1。

我们也许可以将这个结果解释为概率。例如，具有特定 $x_1$ 和 $x_2$ 值的调查参与者，产下男孩的概率为 50%。但是，这个模型也可能产生 $y = 1.1$ 或 $y = -0.1$ 的预测结果，这些都不是有效的概率值。

Logistic 回归将预测结果表示为*优势*（odds）而非概率，避免了上面提到的问题。优势是什么？简单地说，一个事件的“优势”就是该事件发生概率与不发生概率的比值。

因此，如果我认为自己支持的球队有 75% 的获胜几率，那么就可以说他们的优势是 3:1，因此获胜几率是失败几率的 3 倍。

优势和概率是同一信息的不同表达。给定一个概率，可以用如下公式计算优势：

```
o = p / (1-p)
```

给定优势，可以用如下公式转换计算概率：

```
p = o / (o+1)
```

Logistic 回归基于如下模型：

$$\log o=\beta_0+\beta_1 x_1+\beta_2 x_2+\varepsilon$$

其中 $o$ 为特定结果的优势。在预测新生儿性别的例子中，$o$ 就是生男孩的优势。

假设已经估计出参数 $\beta_0$、$\beta_1$ 和 $\beta_2$（稍后会进行解释），并给定 $x_1$ 和 $x_2$ 的值，那么可以计算得到 $\log o$ 的预测值，并转换为概率。

```
o = np.exp(log_o)
p = o / (o+1)
```

因此，在办公室赌局中，我们可以计算出生男孩的预测概率。但是，如何进行参数估计呢？

## 11.7　估计参数

与线性回归不同，Logistic 回归没有闭式解，因此我们只能猜测一个初始解，然后逐步进行优化。

常用的优化目标是找到最大似然估计（maximum-likelihood estimate，MLE），即最大化数据似然的一组参数。例如，假设有如下数据：

```
>>> y = np.array([0, 1, 0, 1])
>>> x1 = np.array([0, 0, 0, 1])
>>> x2 = np.array([0, 1, 1, 1])
```

一开始我们可以估计 $\beta_0=-1.5$，$\beta_1=2.8$，$\beta_2=1.1$。

```
>>> beta = [-1.5, 2.8, 1.1]
```

然后对每个数据行计算 `log_o`。

```
>>> log_o = beta[0] + beta[1] * x1 + beta[2] * x2
[-1.5 -0.4 -0.4  2.4]
```

并将结果从优势对数值转换为概率。

```
>>> o = np.exp(log_o)
[  0.223   0.670   0.670  11.02 ]

>>> p = o / (o+1)
[ 0.182  0.401  0.401  0.916 ]
```

请注意：当 `log_o` 大于 0 时，`o` 大于 1，`p` 大于 0.5。

当 `y==1` 时，一个结果的似然为 `p`；当 `y==0` 时，结果的似然为 `1-p`。例如，如果生男孩的概率为 0.8，结果为男孩的似然就是 0.8；如果结果为女孩，似然则为 0.2。计算方法如下：

```
>>> likes = y * p + (1-y) * (1-p)
[ 0.817  0.401  0.598  0.916 ]
```

数据的总似然为 `likes` 值的乘积：

```
>>> like = np.prod(likes)
0.18
```

对于这些 `beta` 值，数据的似然为 0.18。Logistic 回归的目标是找到使似然最大的参数。为此，大多数的统计软件包都使用类似 Newton 迭代法的解法（参见 https://en.wikipedia.org/wiki/Logistic_regression#Model_fitting）。

## 11.8 实现

StatsModels 提供一个名为 `logit` 的 Logistic 回归实现，这个方法实现将概率转换为优势对数值的函数，因而得名。为了演示其用法，我将寻找影响性别比率的变量。

我仍然加载全国家庭增长调查数据，选择妊娠时间超过 30 周的记录。

```
live, firsts, others = first.MakeFrames()
df = live[live.prglngth>30]
```

`logit` 要求因变量为双值（而非布尔型），因此我创建了一个新列 `boy`，用 `astype(int)` 将布尔值转换为整型双值。

```
df['boy'] = (df.babysex==1).astype(int)
```

已知影响性别比率的因素有：父母年龄、出生排行、种族以及社会地位。Logistic 回归可以检查这些效应是否出现在全国家庭增长调查数据中。首先，从母亲年龄开始。

```
import statsmodels.formula.api as smf

model = smf.logit('boy ~ agepreg', data=df)
results = model.fit()
SummarizeResults(results)
```

`logit` 使用的参数与 `ols` 相同，即 Patsy 语法书写的一个公式，以及一个 DataFrame。`logit` 的结果为一个代表模型的 Logit 对象。Logit 对象的 `endog` 属性包含内生变量（endogenous variable），即因变量，`exog` 属性包含外生变量（exogenous variable），即解释变量。这些变量都是 NumPy 数组，有时转换为 DataFrame 会更方便。

```
endog = pandas.DataFrame(model.endog, columns=[model.endog_names])
exog = pandas.DataFrame(model.exog, columns=model.exog_names)
```

`model.fit` 的结果为 BinaryResults 对象，这个对象与 `ols` 生成的 RegressionResult 对象相似。结果的汇总信息如下：

```
Intercept   0.00579    (0.953)
agepreg     0.00105    (0.783)
R^2 6.144e-06
```

`agepreg` 的参数为正，说明年龄较大的母亲生男孩的可能性更大，但 $p$ 值为 0.783，说明这一直观效应很可能是偶然产生的。

决定系数 $R^2$ 不适用于 Logistic 回归，但有几个可作为“伪 $R^2$ 值”的度量。这些值可以用于进行模型的比较。例如，下面的模型包含几个可能与性别比率相关的因素。

```
formula = 'boy ~ agepreg + hpagelb + birthord + C(race)'
model = smf.logit(formula, data=df)
results = model.fit()
```

除了母亲的年龄，这个模型还使用了孩子出生时父亲的年龄（`hpagelb`）、孩子的出生排行（`birthord`），以及种族作为分类变量。这个模型的拟合结果如下：

```
Intercept      -0.0301      (0.772)
C(race)[T.2]   -0.0224      (0.66)
C(race)[T.3]   -0.000457    (0.996)
agepreg        -0.00267     (0.629)
hpagelb         0.0047      (0.266)
birthord        0.00501     (0.821)
R^2 0.000144
```

这些估计参数都不是统计显著的。结果中的伪 $R^2$ 值略高，但也可能是偶然导致的。

## 11.9　准确度

在办公室赌局中，我们最感兴趣的是模型的准确度：与随机结果相比，成功预测的次数。

在全国家庭增长调查数据中，新生儿男孩比女孩多，因此最简单的预测策略是每次都猜“男孩”。这个策略的准确度是男孩所占的比例。

```
actual = endog['boy']
baseline = actual.mean()
```

actual 编码为双值整数，因此其均值就是男孩的比例，为 0.507。

计算模型准确度的方法如下：

```
predict = (results.predict() >= 0.5)
true_pos = predict * actual
true_neg = (1 - predict) * (1 - actual)
```

results.predict 返回一个包含概率的 NumPy 数组，四舍五入为 0 或 1。如果预测为男孩并且结果正确，那么 predict 乘以 actual 就得到 1，否则为 0。因此，ture_pos 说明“真且正确”。

同样，ture_neg 代表猜测女孩并结果正确。模型的准确度就是正确猜测的比例。

```
acc = (sum(true_pos) + sum(true_neg)) / len(actual)
```

得到的结果为 0.512，比基线 0.507 略高。但是，这个结果不能太当真。我们使用同样的数据构建和检验这个模型，因此，这个模型可能对新数据并不具有预测能力。

无论如何，还是让我们使用这个模型来进行办公室赌局的预测。假设这位怀孕的同事 35 岁，白人，丈夫 39 岁，这是他们的第 3 个孩子。

```
columns = ['agepreg', 'hpagelb', 'birthord', 'race']
new = pandas.DataFrame([[35, 39, 3, 2]], columns=columns)
y = results.predict(new)
```

调用 results.predict 进行新的预测，必须构建一个 DataFrame，为模型中的每个变量提供一列。这个预测的结果是 0.52，因此你应该猜“男孩”。但是，即便这个模型真的能提高获胜机会，区别也是非常微小的。

## 11.10　练习

本章练习的参考答案位于 chap11soln.ipynb 中。

- 练习 11.1

假设你的一位同事怀孕待产，办公室发起了一个赌局，预测孩子的出生日期。假定在产妇妊娠的第 30 周下注，可以使用什么变量进行最佳预测？你只能使用在孩子出生前就已知的变量，而且是大家都能得到的信息。

- 练习 11.2

Trivers-Willard 假设认为，很多哺乳动物的性别比率取决于“母方条件”，即如母亲的年龄、体型、健康状况以及社会地位等因素。请参见 https://en.wikipedia.org/wiki/Trivers-Willard_hypothesis。

一些研究显示这一效应在人类中也存在，但结果并不明晰。本章检验了与这些因素相关的一些变量，但并未发现这些变量对性别比率产生了任何统计显著的效应。

作为练习，请使用数据挖掘方法，检验妊娠数据文件和调查参与者文件中的其他变量。你能找到具有显著效应的因素吗？

- 练习 11.3

如果要进行的预测是一个计数值，则可以使用 Poisson 回归。StatsModels 的 `poisson` 函数实现了 Poisson 回归，用法与 `ols` 和 `logit` 相同。作为练习，请使用 `poisson` 函数预测一位妇女生育了几个孩子。在全国家庭增长调查数据集中，这个变量是 `numbabes`。

假设有一位妇女，35 岁，黑人，大学毕业，年均家庭收入超过 75 000 美元，请预测这位妇女已经生育了几个孩子。

- 练习 11.4

如果要预测的是类别，则可以使用多项式 Logistic 回归。StatsModels 的 `mnlogit` 函数实现了多项式 Logistic 回归。作为练习，请使用 `mnlogit` 预测一位妇女的婚姻状态：已婚、同居、守寡、离异、分居或未婚。在全国家庭增长调查数据集中，记录婚姻状况的编码变量为 `rmarital`。

假设有一位妇女，25 岁，白人，高中毕业，年均家庭收入约为 45 000 美元，请预测这位妇女已婚的概率、同居的概率，等等。

## 11.11 术语

- 回归（regression）
  估计模型参数以拟合数据的几个相关过程之一。
- 因变量（dependent variable）
  回归模型中，希望进行预测的变量，也称为内生变量。
- 解释变量（explanatory variable）
  用于预测或解释因变量的变量，也称为自变量或外生变量。
- 简单回归（simple regression）
  只有一个因变量和一个解释变量的回归。
- 多重回归（multiple regression）
  有多个解释变量，但只有一个因变量的回归。

- 线性回归（linear regression）
  基于线性模型的回归。
- 普通最小二乘法（ordinary least square）
  通过最小化残差的均方误差，进行参数估计的线性回归。
- 伪关系（spurious relationship）
  两个变量间的关系，由统计结果造成，或由模型之外但与两个变量都相关的因素导致。
- 控制变量（control variable）
  回归中的变量，用于消除或“控制”伪关系。
- 代理变量（proxy variable）
  因与其他因素相关，成为该因素的代理，从而对回归模型产生间接影响的变量。
- 分类变量（categorical variable）
  一种变量，取值为一组无序的离散值之一。
- 连接（join）
  使用一个键值匹配数据行，结合两个 DataFrame 中的操作。
- 数据挖掘（data mining）
  通过检验大量模型寻找变量关系的方法。
- Logistic 回归（logistic regression）
  因变量为布尔型时使用的一种回归。
- Poisson 回归（Poisson regression）
  因变量为非负整数（通常为一个计数值）时使用的一种回归。
- 优势（odds）
  概率 $p$ 的另一种表示方法，即概率与其补值的比率，$p/(1-p)$。

# 第12章

# 时间序列分析

时间序列（time series）是来自随时间变化的系统的一系列度量。展示全球平均温度变化的“曲棍球棒图”就是一个时间序列的著名示例（参见 https://en.wikipedia.org/wiki/Hockey_stick_graph）。

本章使用的示例来自 Zachary M. Jones。Jones 是一位研究美国大麻黑市的政治学者（http://zmjones.com/marijuana），他的数据来自一个名为“草价”（price of weed）的网站，这个网站请参与者报告大麻交易的价格、数量、质量和地点，以此收集市场信息（http://www.priceofweed.com/）。Jones 的研究目的是调查像大麻合法化这样的政策性决定会对市场产生何种影响。这个研究项目用数据解答重要的政策问题，如药品政策问题，这让我产生了很大的兴趣。

我希望你对本章内容感兴趣，但还是要借此机会重申对数据分析保持专业态度的重要性。药品是否非法，哪些药品应当属于非法，这是很重要而又难以回答的公共政策问题，人们应当基于诚实准确的数据进行决策。

本章代码位于 timeseries.py 中。前言介绍了如何下载和使用本书代码。

## 12.1 导入和清洗数据

从 Jones 先生的网站下载的数据存放在本书代码库中。下面的代码将这些数据读取到一个 pandas DataFrame 中：

```
transactions = pandas.read_csv('mj-clean.csv', parse_dates=[5])
```

`parse_dates` 参数告诉 `read_csv` 方法将第 5 列中的值作为日期读取，并转换为 NumPy 的 `datetime64` 对象。

这个 DataFrame 中每一行都代表一次交易，具有如下列。

- city
  字符串，代表城市名。
- state
  州名的两字母缩写。
- price
  交易价格，单位为美元。
- amount
  购买量，单位为克。
- quality
  由购买者汇报的货品质量，值为高、中或低。
- date
  汇报日期，这个日期应该在购买日期后不久。
- ppg
  每克价格，单位为美元。
- state.name
  字符串，代表州名。
- lat
  交易发生地点的大致纬度，由城市名获得。
- lon
  交易发生地点的大致经度。

每次交易都是一个时间事件，因此这个数据集可以看成一个事件序列。但是，这些事件发生的时间并不均匀，每天汇报的交易数从 0 到数百不等。很多分析时间序列的方法要求度量是均匀分布的，或者度量均匀分布至少可以使分析更为简单。

为了演示这些方法，我将这个数据集按照汇报的质量分组，并计算每克的日均价格，将每组数据转换为均匀分布的序列。

```
def GroupByQualityAndDay(transactions):
    groups = transactions.groupby('quality')
```

```
    dailies = {}
    for name, group in groups:
        dailies[name] = GroupByDay(group)

    return dailies
```

groupyb 是 DataFrame 的方法，返回一个 GroupBy 对象 groups。groups 用在 for 循环中，可以遍历各组的名字及其 DataFrame。由于 quality 列的取值为 low、medium 和 high，因此得到的 3 个组名，即为 low、medium 和 high。

循环遍历这些组，并调用 GroupByDay 方法，计算日平均价格，返回一个新的 DataFrame。

```
def GroupByDay(transactions, func=np.mean):
    grouped = transactions[['date', 'ppg']].groupby('date')
    daily = grouped.aggregate(func)

    daily['date'] = daily.index
    start = daily.date[0]
    one_year = np.timedelta64(1, 'Y')
    daily['years'] = (daily.date - start) / one_year

    return daily
```

参数 transaction 是包含 date 和 ppg 列的 DataFrame。代码选择这两列，然后按 date 分组。

得到的 grouped 是一个映射，将每个日期对应到包含该日期汇报价格的 DataFrame。aggregate 是 GroupBy 的方法，遍历所有的组，并对组中每列都执行一个函数。在 GroupByDay 方法中，grouped 组中只有一列 ppg。因此，aggregate 生成的 DataFrame 只有一列 ppg，每个日期有一行数据。

这些 DataFrame 中的日期存储为 NumPy datetime64 对象，为 64 位整数，包含该日期的十亿分之一秒值。对于稍后介绍的一些分析方法，使用人们易于理解的时间单位（例如年）更加方便。因此，GroupByDay 通过复制 index 增加一个新列 date，然后增加一个 years 列，将距第一次交易日期的年数存储为浮点数。

GroupByDay 方法得到的 DataFrame 包含列 ppg、date 和 years。

## 12.2 绘制图形

GroupByQualityAndDay 得到的结果是从每个质量级别到日均价格 DataFrame 的映射。绘制这 3 个时间序列的代码如下：

```
thinkplot.PrePlot(rows=3)
for i, (name, daily) in enumerate(dailies.items()):
    thinkplot.SubPlot(i+1)
```

```
        title = 'price per gram ($)' if i==0 else ''
        thinkplot.Config(ylim=[0, 20], title=title)
        thinkplot.Scatter(daily.index, daily.ppg, s=10, label=name)
        if i == 2:
            pyplot.xticks(rotation=30)
        else:
            thinkplot.Config(xticks=[])
```

PrePlot 使用参数 rows=3，说明要在 3 行绘制 3 个图形。循环遍历 dailies 中的 DataFrame，为每个 DataFrame 绘制一个散点图。绘制时间序列时，一种常见做法是在点之间连上线段。但本示例 dailies 中 DataFrame 的数据点很多，而且价格变化很大，因此添加线段并没有什么用处。

由于横轴上的标签是日期，因此代码中使用 pyplot.xticks，此外将刻度旋转 30 度，能使其更加易读。

图 12-1 展示了代码运行的结果。这些图形有个明显的特点：2013 年 11 月附近有一个缺口。这可能是因为那段时间没有进行数据收集，或者无法获得数据。我们稍后将讨论处理这种缺失数据的方法。

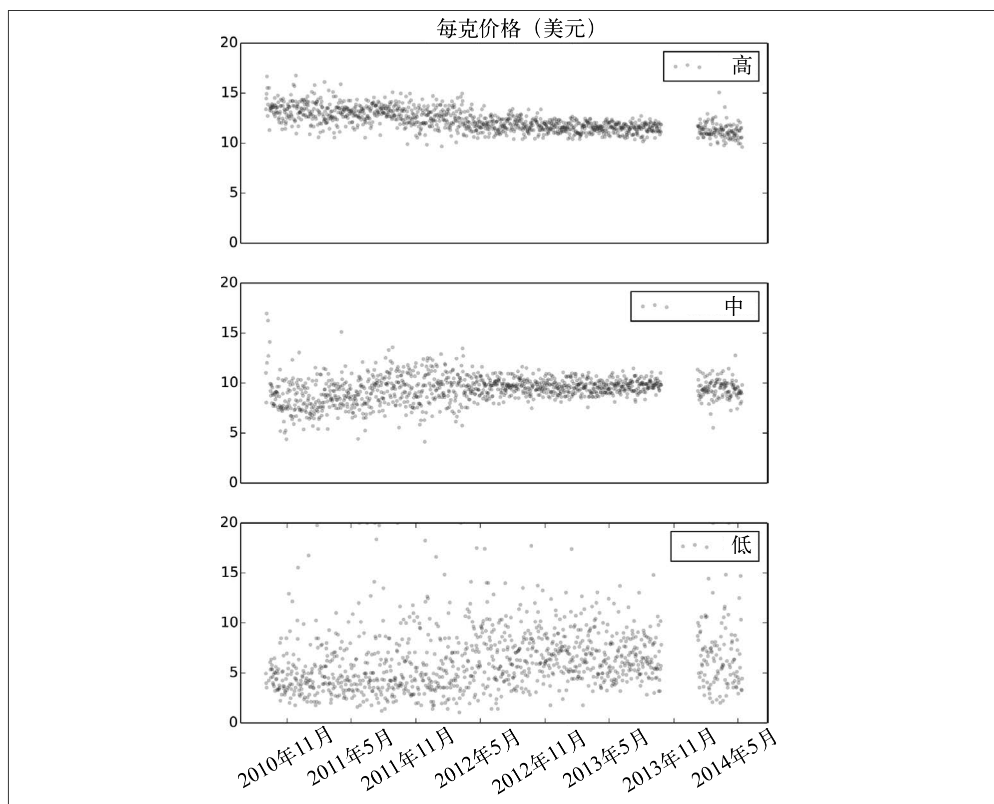



图 12-1：高、中、低质量大麻的每克每日价格时间序列

从图中我们可以看到，这段时间内，高质量大麻的价格似乎在降低，中等质量大麻的价格在提高。低质量大麻的价格似乎也在提高，但其价格变化过大，很难判断趋势。请记住，大麻质量的数据是由志愿者提供的，因此，这些随时间变化的趋势可能反映了参与者判断标准的变化。

## 12.3 线性回归

虽然时间序列分析有专门的方法，但是对于很多问题，简单的方法是首先使用通用的工具，如线性回归。下面的函数以每日价格的 DataFrame 为参数，计算一个最小二乘法拟合，返回使用 StatsModels 包得到的模型和结果对象。

```
def RunLinearModel(daily):
    model = smf.ols('ppg ~ years', data=daily)
    results = model.fit()
    return model, results
```

然后，我们可以遍历 `dailies` 中的 DataFrame，为每个 DataFrame 拟合模型。

```
for name, daily in dailies.items():
    model, results = RunLinearModel(daily)
    print(name)
    regression.SummarizeResults(results)
```

结果如下：

| 质量 | 截距 | 斜率 | $R^2$ |
| --- | --- | --- | --- |
| 高 | 13.450 | −0.708 | 0.444 |
| 中 | 8.879 | 0.283 | 0.050 |
| 低 | 5.362 | 0.568 | 0.030 |

估计所得斜率说明，在观测区间内，高质量大麻的价格每年下降约 71%；中等质量大麻的价格每年上涨约 28%；低质量大麻价格每年上涨 57%。这些估计值都是统计显著的，$p$ 值很小。

高质量大麻的 $R^2$ 值为 0.44，说明以时间为解释变量，可以解释所观测到的价格变化的 44%。其他两类的价格变化较小，价格可变性较高，因此 $R^2$ 值较小（但仍然是统计显著的）。

下面的代码绘制了观测到的价格以及拟合值：

```
def PlotFittedValues(model, results, label=''):
    years = model.exog[:,1]
    values = model.endog
    thinkplot.Scatter(years, values, s=15, label=label)
    thinkplot.Plot(years, results.fittedvalues, label='model')
```

如 11.8 节介绍的，`model` 包含 `exog` 和 `endog`，这两个 NumPy 数据包含外生变量（即解释变量），以及内生变量（即因变量）。

`PlotFittedValues` 绘制数据点的散点图以及拟合值的线图。图 12-2 展示了高质量大麻的绘制结果。

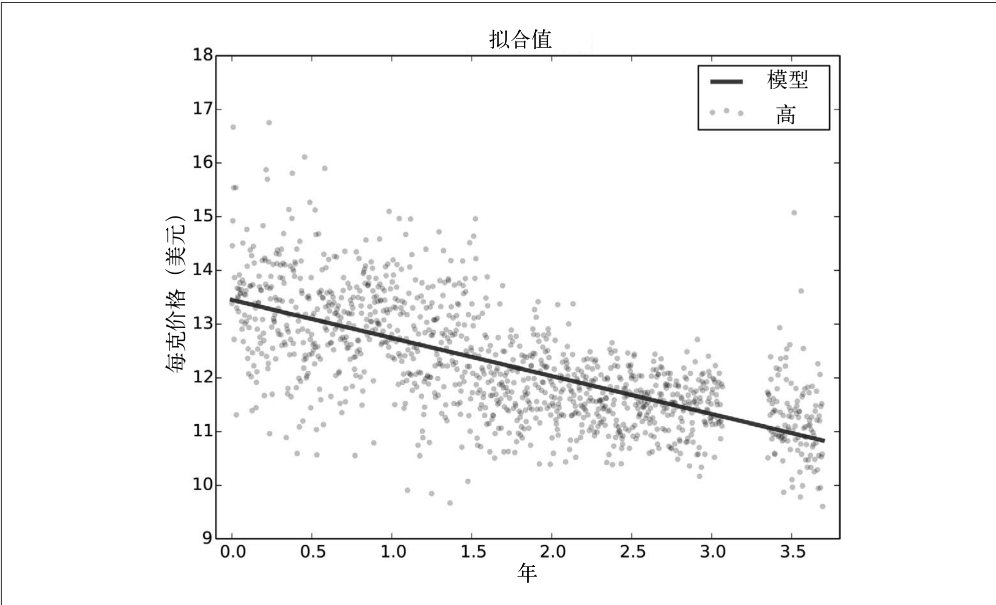



图 12-2：高质量大麻每克每日价格的时间序列以及线性最小二乘法拟合

图中的模型似乎很好地拟合了数据，然而，线性回归并不是拟合这种数据的最适宜方法，原因如下。

- 首先，预期长期趋势为线性的或符合其他简单函数，这是没有依据的。通常，价格由供需关系决定，而供给和需求都会随着时间发生不可预期的变化。
- 其次，线性回归模型对所有数据使用同样的权重，不管是过去的还是近期数据。在进行预测时，近期数据也许应该得到更多的权重。
- 最后，线性回归假设残差是无关的噪音。而在时间序列数据中，相邻数据是相关的，这项假设通常不能成立。

下一节将介绍更适合时间序列数据的分析方法。

# 12.4　移动平均值

大部分时间序列分析基于的建模假设都认为，观测序列是三部分的总和。

- *趋势*

  描述持续变化的一个平滑函数。

- *季节性*

  周期性的变化，可能包括每日、每周、每月或每年周期。

- *噪音*

  长期趋势周围的随机变化。

如前一节介绍的，回归是从一个序列中获取趋势的方法。但是，如果这个趋势不是一个简单函数，那么一个很好的方法是*移动平均值*（moving average）。移动平均值将序列分为相互重叠的区域（称为*窗口*，即 window），计算每个窗口的平均值。

最简单的移动平均值是*滚动均值*（rolling mean），滚动均值计算每个窗口中值的均值。例如，如果窗口大小为 3，那么滚动均值就计算值 0 到值 2 的均值，值 1 到值 3 的均值，值 2 到值 4 的均值，依此类推。

pandas 提供 `rolling_mean` 方法，参数为 Series 和窗口大小，返回一个新的 Series。

```
>>> series = np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> pandas.rolling_mean(series, 3)
array([ nan, nan,    1,   2,   3,   4,   5,   6,   7,   8])
```

返回 Series 的前两个值为 `nan`，随后的值是前三个元素 0、1 和 2 的均值，再往后是 1、2 和 3 的均值，以此类推。

在对大麻数据使用 `rolling_mean` 方法之前，必须首先处理缺失数据。在观测时间中，有几天缺少一个或多个质量分类的报告交易，2013 年有一段时间没有数据。

我们目前使用的 DataFrame 缺少这些数据，索引跳过了没有数据的日期。为了随后的分析，我们需要明确标明这些缺失数据，为此，可以对 DataFrame 进行“重建索引”。

```
dates = pandas.date_range(daily.index.min(), daily.index.max())
reindexed = daily.reindex(dates)
```

第一行代码计算一个日期范围，其中包含从观测时间开始到结束的每一天。第二行代码创建一个新的 DataFrame，其中包含 `daily` 中的所有数据，日期范围中的每一天都对应一行，缺失数据用 `nan` 填充。

下面的代码绘制了滚动均值：

```
roll_mean = pandas.rolling_mean(reindexed.ppg, 30)
thinkplot.Plot(roll_mean.index, roll_mean)
```

窗口大小为 30，因此 `roll_mean` 中的每个值都是 `reindexed.ppg` 中 30 个值的均值。

图 12-3（左）展示了代码运行结果。滚动均值似乎很好地对噪音进行了平滑处理，提取出了趋势。前 29 个值为 `nan`，缺失数据的部分也有 29 个 `nan`。这些空缺可以使用一些方法进行填充，但不处理也无关紧要。

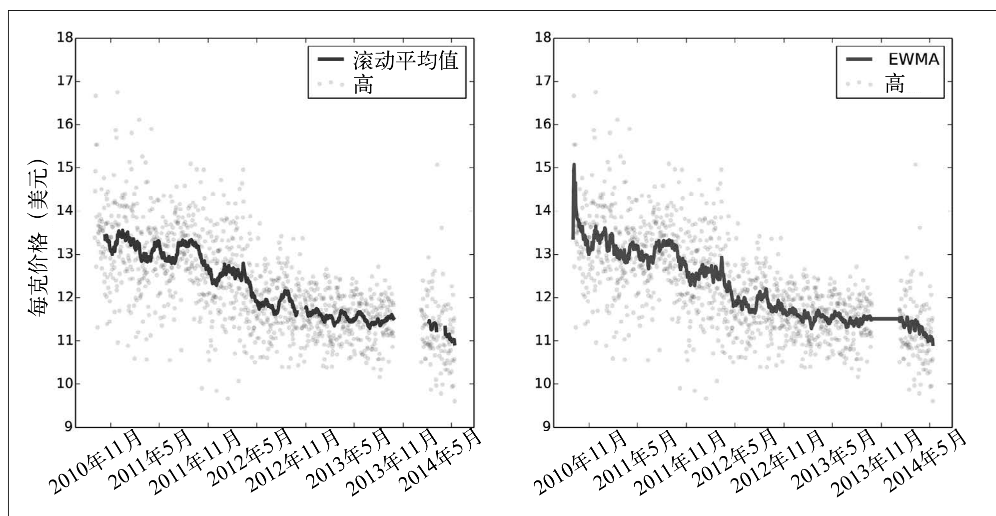



图 12-3：每日价格、滚动均值（左）和指数权重移动平均（右）

另一种移动平均值是*指数权重移动平均*（exponentially-weighted moving average，EWMA）。指数权重移动平均有两个优点：首先，如其名所示，指数权重移动平均计算加权平均值，最近的值具有最高的权重，之前值的权重指数级降低；第二，pandas 的 EWMA 实现可以较好地处理缺失值。

```
ewma = pandas.ewma(reindexed.ppg, span=30)
thinkplot.Plot(ewma.index, ewma)
```

参数 `span` 大致等同于移动平均值的窗口大小。这个参数控制权重降低的速度，因此决定了对每个平均值产生不可忽略作用的值点数目。

图 12-3（右）展示了同样数据使用 EWMA 方法的分析结果。在有数据的部分，右侧结果与左侧类似，但是右侧图形没有缺失值，从而更容易使用。时间序列开始部分的值基于的数据点较少，因此噪声较大。

## 12.5　缺失值

得到时间序列趋势后，下一步就要研究其季节性，即周期性行为。基于人类行为的时间序列数据经常展现出每日、每周、每月或每年的周期。下一节将介绍检验季节性的方法，但

这些方法会受到缺失数据的影响，因此我们需要首先解决缺失值的问题。

填充缺失数据，一个简单常用的方法是使用移动平均值。Series 的方法 `fillna` 就实现了这一功能。

```
reindexed.ppg.fillna(ewma, inplace=True)
```

`fillna` 将 `reindexed.ppg` 中的 `nan` 替换为 `ewma` 中的相应值。`inplace` 参数告诉 `fillna` 直接修改现有的 Series，而不是创建新的 Series。

这一方法的缺点是弱化了序列中的噪音，这一问题可以通过添加重抽样的残差解决。

```
resid = (reindexed.ppg - ewma).dropna()
fake_data = ewma + thinkstats2.Resample(resid, len(reindexed))
reindexed.ppg.fillna(fake_data, inplace=True)
```

`resid` 包含残差，但不包括 `ppg` 为 `nan` 的日期。`fake_data` 包含移动平均值的结果与残差的一个随机样本的和。最后，`finllna` 将 `nan` 替换为 `fake_data` 中的值。

图 12-4 展示了填充数据后的结果。填充数据看起来与实际数据非常相似。由于重抽样的残差是随机的，因此每次填充的结果都不同。我们稍后将讨论如何描述由缺失数据导致的误差。

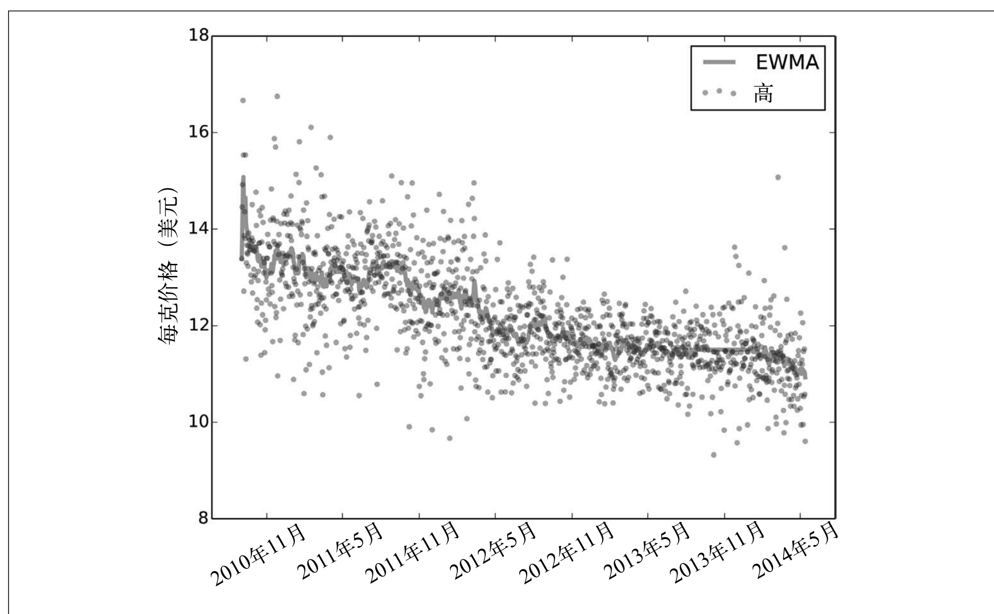



图 12-4：填充数据后的每日价格

## 12.6 序列相关

随着价格每日变化，你可能会期望看到一些模式。如果周一价格很高，可能随后几天都会较高；如果价格很低，可能会保持低位。在这种模式中，每个值都与序列中的下一个值相关，因此称为*序列相关*（serial correlation）。

要计算序列相关，我们可以将时间序列移动一个称为*滞后*（lag）间隔，然后计算移动后的序列与原序列的相关性。

```
def SerialCorr(series, lag=1):
    xs = series[lag:]
    ys = series.shift(lag)[lag:]
    corr = thinkstats2.Corr(xs, ys)
    return corr
```

移动后的序列中，前 `lag` 个值都为 `nan`，因此代码在计算 `Corr` 之前会将前 `lag` 个值移除。

如果使用原始的价格数据，以 `lag` 为 1 运行 `SerialCorr`，得到的结果为：高质量大麻的序列相关为 0.48，中等质量的为 0.16，低质量的为 0.10。任何具有长期趋势的时间序列都具有很强的序列相关。例如，如果价格走低，序列前半部分值会高于均值，而后半部分值低于均值。

更有意思的是，减去这个趋势后，检验这种相关是否依然存在。例如，我们可以算出这个 EWMA 的残差，然后计算其序列相关。

```
ewma = pandas.ewma(reindexed.ppg, span=30)
resid = reindexed.ppg - ewma
corr = SerialCorr(resid, 1)
```

在 `lag=1` 时，减去趋势后数据的序列相关为：高质量为 −0.022，中等质量为 −0.015，低质量为 0.036。这些值都很小，说明序列中的一日序列相关很小或不存在。

使用不同的滞后值，可以检验序列的每周、每月和每年的季节性特征。具体结果如下：

| 滞后 | 高质量 | 中等质量 | 低质量 |
|---|---|---|---|
| 1 | −0.029 | −0.014 | 0.034 |
| 7 | 0.02 | −0.042 | −0.0097 |
| 30 | 0.014 | −0.0064 | −0.013 |
| 365 | 0.045 | 0.015 | 0.033 |

下一节将检验这些相关是否统计显著（结果不是），现在我们可以暂时认为，这些序列没有显著的季节性模式，至少在使用上述滞后值时没有。

## 12.7 自相关

如果你认为一个序列可能具有一定的序列相关，但却不知道该使用哪些滞后值进行检验，那就可以全部检验一遍！自相关函数（autocorrelation function）将滞后值映射到使用该值得到的序列相关。“自相关”是序列相关的另一个名字，常用于滞后值不为 1 时。

11.1 节使用 StatsModels 进行线性回归。这个软件包也提供时间序列分析函数，其中有计算自相关函数的 `acf`。

```
import statsmodels.tsa.stattools as smtsa
acf = smtsa.acf(filled.resid, nlags=365, unbiased=True)
```

`acf` 使用从 0 到 `nlags` 的滞后值计算序列相关。参数 `unbiased` 为 `True`，告诉 `acf` 要为样本规模校正估计值。`acf` 的结果是一个相关性数组。如果选择高质量大麻的每日价格，抽取滞后为 1、7、30 和 365 的相关值，就可以验证 `acf` 和 `SerialCorr` 得到的结果大致相同。

```
>>> acf[0], acf[1], acf[7], acf[30], acf[365]
1.000, -0.029, 0.020, 0.014, 0.044
```

`lag=0` 时，`acf` 计算的是序列与自身的相关，这个值总是为 1。

图 12-5（左）展示了 `nlags=40` 时，3 种质量分类的自相关函数。图中的灰色区域是不存在自相关时的正态可变性，位于这个区域之外的都是统计显著的，$p$ 值小于 5%。误报率为 5%，因此计算 120 个相关时（3 个时间序列，每个序列有 40 个滞后值），大约会有 6 个点在灰色区域之外。实际上，图中有 7 个点在灰色区域外。我们据此认为，在这些序列中，不存在无法用偶然性解释的自相关。

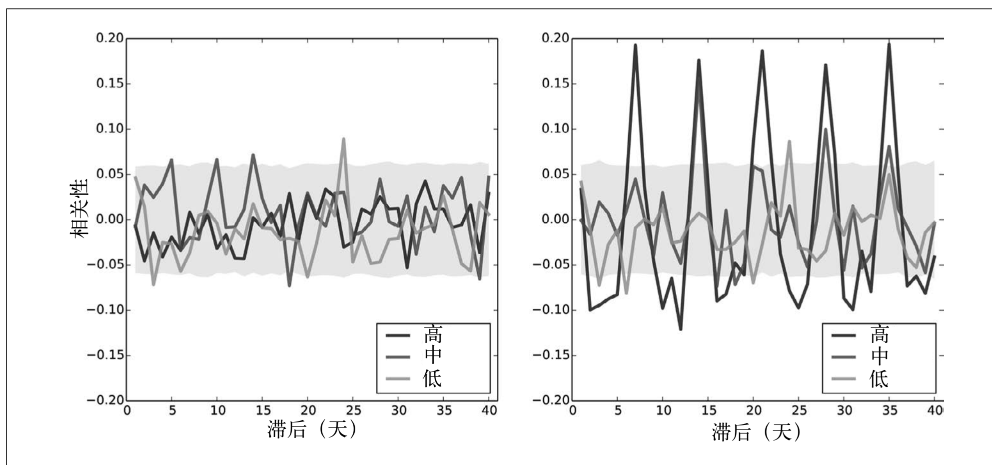



图 12-5：每日价格的自相关函数（左）及模拟的每周季节性的每日价格（右）

灰色区域是通过对残差重抽样计算得到的。具体代码位于 timeseries.py 中，函数名为 SimulateAutocorrelation。

为了展示存在季节性因素的自相关函数，我在数据中加入一个每周的循环进行模拟。假设在周末时大麻的需求量较大，那么价格可能会较高。为了模拟这个效果，我选取了周五或周六的日期，在价格上添加一个随机量，这个随机量选自从 0~2 美元的均匀分布。

```
def AddWeeklySeasonality(daily):
    frisat = (daily.index.dayofweek==4) | (daily.index.dayofweek==5)
    fake = daily.copy()
    fake.ppg[frisat] += np.random.uniform(0, 2, frisat.sum())
    return fake
```

frisat 是一个布尔型 Series，如果日期为周五或周六，则值为 True。fake 是一个新的 DataFrame，从 daily 复制而来，而后对 ppg 添加随机值进行修改。frisat.sum() 是周五和周六日期的总数，也就是需要生成的随机值的数量。

图 12-5（右）展示出添加了模拟季节性的价格自相关函数。正如我们预期的，当滞后为 7 的倍数时，相关性最高。对于高质量和中等质量大麻，新得到的相关是统计显著的。而低质量大麻则不然，因为这一分类的残差最大，必须使用更大的模拟值，才能使效应在噪音中显现出来。

## 12.8　预测

时间序列分析可以用于调查，有时也可用于解释随时间变化的系统行为。时间序列分析还可用于预测。

12.3 节中使用的线性回归可以用于预测。RegressionResults 类提供 predict 方法，以包含解释变量的 DataFrame 为参数，返回一个预测序列。使用方法如下：

```
def GenerateSimplePrediction(results, years):
    n = len(years)
    inter = np.ones(n)
    d = dict(Intercept=inter, years=years)
    predict_df = pandas.DataFrame(d)
    predict = results.predict(predict_df)
    return predict
```

results 是一个 RegressionResults 对象，years 是待预测的时间值序列。这个函数构建一个 DataFrame，将其传给 predict 方法，并返回得到的结果。

如果我们希望得到的只是单一的、最佳推测预测，那么上面的函数就可以满足需求。但是，大部分时候我们还需要对误差进行量化，即希望得知预测结果的准确性如何。

我们需要考虑三种误差来源。

- 抽样误差

  预测基于估计参数，而估计参数依赖样本中的随机变异。如果重复进行实验，估计值会发生变化。

- 随机变异

  即使估计参数是完美的，观测数据也会在长期趋势附近随机变动，这种变异在未来也会持续出现。

- 建模误差

  前面已经有示例证明长期趋势不是线性的，因此，基于线性模型的预测终究会失败。

另一种误差来源于无法预期的未来事件。农产品价格受天气影响，所有的价格都会受政策和法律影响。在本书编撰时，美国已经有两个州宣布大麻合法，还有 20 多个州宣布医用大麻合法。如果更多的州宣布大麻合法化，那么大麻价格很可能会下跌。但是，如果联邦政府要打击大麻交易，价格则可能会上涨。

建模误差和无法预期的未来事件很难量化。而抽样误差和随机变异较容易处理，因此我们先来处理这两种误差。

我依然采用 10.4 节中用过的重抽样方法对抽样误差进行量化。和往常一样，重抽样的目的是使用实际观测来模拟重复进行实验时可能得到的数据。这种模拟基于的假设是，估计参数是正确的，但随机残差可能会不同。实现这种模拟的函数如下：

```
def SimulateResults(daily, iters=101):
    model, results = RunLinearModel(daily)
    fake = daily.copy()

    result_seq = []
    for i in range(iters):
        fake.ppg = results.fittedvalues + Resample(results.resid)
        _, fake_results = RunLinearModel(fake)
        result_seq.append(fake_results)

    return result_seq
```

`daily` 是包含观测价格的 DataFrame，`iters` 是模拟的次数。

`SimulateResults` 使用了 12.3 节中介绍的 `RunLinearModel`，估计观测值的斜率和截距。

在每次循环中，代码对残差进行重抽样，将其附加在拟合值上，生成一个“伪”数据集，然后对伪数据拟合一个线性模型，将结果存储在 RegressionResults 对象中。

下一步是使用模拟结果生成预测：

```
def GeneratePredictions(result_seq, years, add_resid=False):
    n = len(years)
```

```
    d = dict(Intercept=np.ones(n), years=years, years2=years**2)
    predict_df = pandas.DataFrame(d)

    predict_seq = []
    for fake_results in result_seq:
        predict = fake_results.predict(predict_df)
        if add_resid:
            predict += thinkstats2.Resample(fake_results.resid, n)
        predict_seq.append(predict)

    return predict_seq
```

GeneratePredictions 的参数 result_seq 是上一步生成的结果序列；参数 years 是一个浮点数序列，指定待生成预测的时间区间；参数 add_resid 说明是否应该将重取样的残差附加到直线预测结果上。GeneratePredictions 遍历 RegressionResults 序列，并生成一个预测序列。

最后一步是绘制预测结果的 90% 置信区间。

```
def PlotPredictions(daily, years, iters=101, percent=90):
    result_seq = SimulateResults(daily, iters=iters)
    p = (100 - percent) / 2
    percents = p, 100-p

    predict_seq = GeneratePredictions(result_seq, years, True)
    low, high = thinkstats2.PercentileRows(predict_seq, percents)
    thinkplot.FillBetween(years, low, high, alpha=0.3, color='gray')

    predict_seq = GeneratePredictions(result_seq, years, False)
    low, high = thinkstats2.PercentileRows(predict_seq, percents)
    thinkplot.FillBetween(years, low, high, alpha=0.5, color='gray')
```

PlotPredictions 两次调用 GeneratePredictions 方法，一次使用参数 add_resid=True，另一次使用 add_resid=False。代码使用 PercentileRows，选择每一年的第 5 和第 95 百分位秩，在其间绘制一个灰色区域。

图 12-6 展示了代码运行的结果。图中深灰色区域代表取样误差的 90% 置信区间，即由取样导致的估计斜率和截距不确定性。

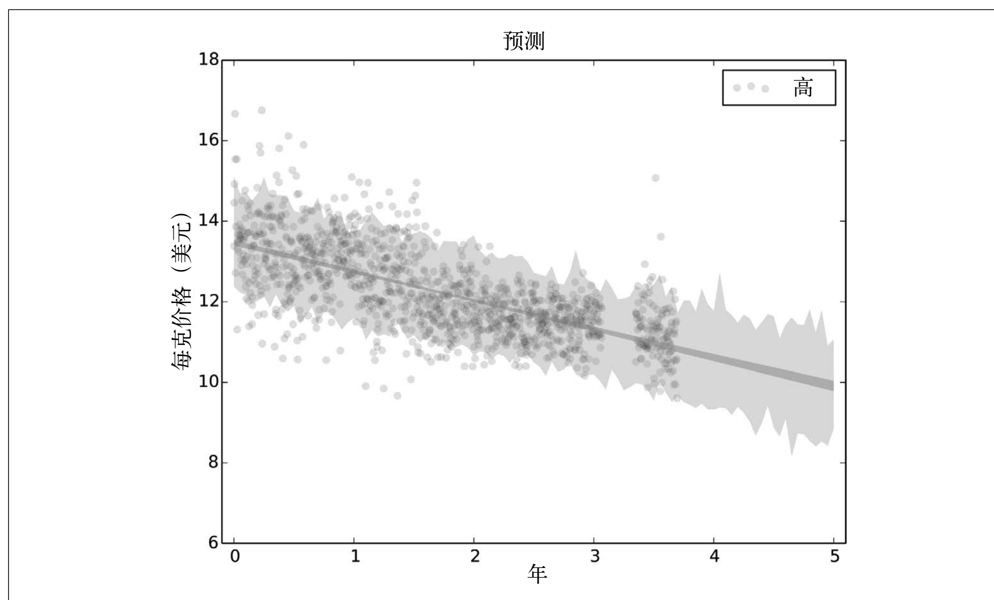



**图 12-6：基于线性拟合的预测，展示了由抽样误差和预测误差导致的变异**

图中浅灰色区域展示了预测误差的 90% 置信区间，预测误差是取样误差和随机变异的和。

这些区域对取样误差和随机变异进行了量化，但并不包括建模误差。建模误差通常很难量化，但在我们的示例中，至少有一个误差来源可以处理，即无法预测的外部事件。

回归模型基于的假设是，系统是*平稳的*（stationary），即模型参数不会随着时间发生变化。具体来说，回归模型假设模型的斜率和截距是常数，残差分布也不变。

但是在图 12-3 的移动平均值图中，斜率在观测区间中似乎至少变化了一次，而且前半部分的残差方差似乎也比后半部分大。

因此，我们得到的参数依赖观测区间。为了检验这一现象对预测结果的影响，我们可以对 `SimulateResults` 进行扩展，使用具有不同开始和结束日期的观测区间进行拟合，具体实现请参见 timeseries.py。

图 12-7 展示了中等质量分类的预测结果。图中最浅的灰色区域代表各种误差的置信区间，包括取样误差、随机变异和观测区间变化。

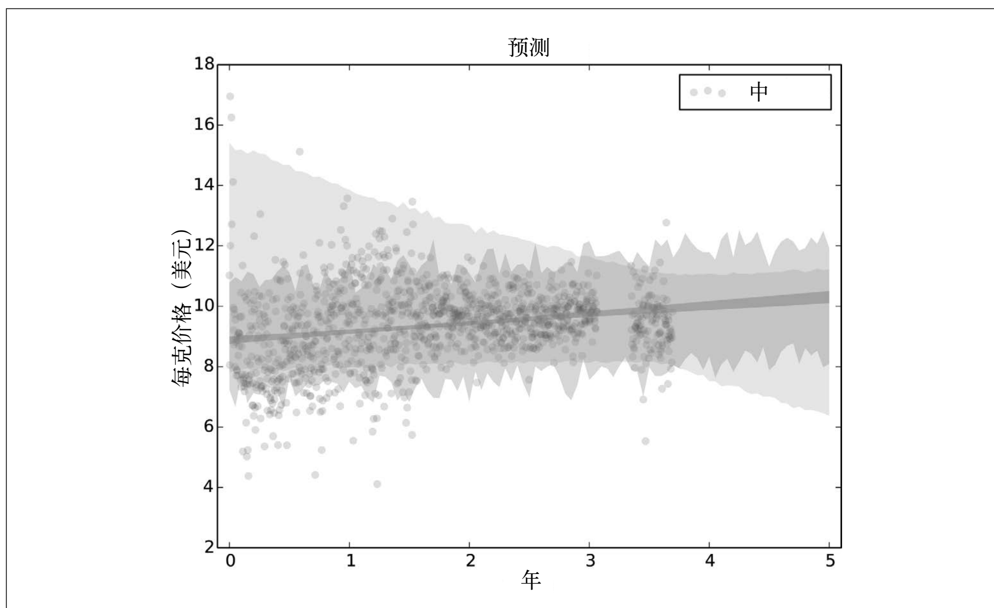



图 12-7：基于线性拟合的预测，展示了由观测间隔导致的变异

基于整个区间的模型的斜率为正数，说明价格在上涨。而最近的区间显示出价格有下跌迹象，因此基于最近数据的模型斜率为负数。因此，最宽的预测间隔包含了下一年价格下跌的可能性。

## 12.9 参考书目

时间序列分析是一个很大的课题，这一章只触及皮毛而已。进行时间序列数据处理的一个重要工具是自回归。本章没有介绍自回归，主要是因为这个工具不适合我们所用的示例数据。

但是，学完本章之后，你完全可以自己学习自回归。我推荐 Philipp Janert 的 *Data Analysis with Open Source Tools*，O’Reilly Media，2011。这本书中关于时间序列分析的章节很好地补充了本章的内容。

## 12.10 练习

本章练习的参考答案位于 chap12soln.py 中。

- 练习 12.1

本章使用了线性模型，因其是线性的，所以具有明显的缺点。我们没有理由预期价格随时间线性变化。我们可以使用 11.3 节中的方法，添加一个二次函数，从而增加模型的灵活性。

请使用一个二次模型，对每日价格时间序列进行拟合，并使用该模型生成预测结果。你需要编写一个新版本的 `RunLinearModel` 函数，运行这个二次模型，然后即可重用 timeseries.py 中的代码生成预测结果。

- 练习 12.2

请编写一个名为 `SerialCorrelationTest` 的类，该类继承 9.2 节中的 `HypothesisTest`。这个类应当使用一个 Series 和一个滞后值，用指定的滞后值计算该 Series 的序列相关，然后计算观测相关的 $p$ 值。

请使用这个类检验原始价格数据中的序列相关是否为统计显著的，并检验线性模型的残差以及（前一个练习生成的）二次模型的残差。

- 练习 12.3

有几种方法可以扩展 EWMA 模型生成预测结果，最简单的一种为：

(1) 计算时间序列的 EWMA，取最后一点为截距 `inter`；
(2) 计算时间序列中相邻元素差值的 EWMA，取最后一点为斜率 `slope`；
(3) 计算 `inter + slope * dt` 预测未来值，其中 `dt` 为预测时间与最后一个观测时间的差值。

请使用这种方法，生成最后一个观测后一年的预测值。提示如下：

- 运行分析之前，使用 `timeseries.FillMissing` 填充缺失值，这样可使相邻元素间的时间差一致；
- 使用 `Series.diff` 计算相邻元素的差值；
- 使用 `reindex` 将 DataFrame 的索引扩展到未来时间；
- 使用 `fillna` 将预测值写入 DataFrame 中。

# 12.11 术语

- 时间序列（time series）
  一个数据集，其中每个值都与一个时间戳相关，通常为一系列测量值及其收集时间。

- 窗口（window）
  时间序列中的一列连续值，经常用于计算移动平均值。

- 移动平均值（moving average）
  用于估计时间序列的潜在趋势的统计量之一，通过计算一些列重叠窗口的（某种）平均值得到。

- 滚动均值（rolling mean）
  基于每个窗口均值的一种移动平均值。

- 指数加权移动平均（exponentially-weighted moving avearage，EWMA）
  一种基于加权均值的移动平均值，最近的值具有最高的权重，早期值的权重按指数级降低。

- span：
  EWMA 的一个参数，用于控制权重降低的速度。

- 序列相关（serial correlation）
  一个时间序列和它自身的一个移动或滞后版本间的相关。

- 滞后（lag）
  序列相关或自相关中数据移动的大小。

- 自相关（autocorrelation）
  一个更为通用的术语，描述使用任意滞后值的序列相关。

- 自相关函数（autocorrelation function）
  将滞后值映射到序列相关的函数。

- 平稳（stationary）
  如果一个模型的参数和残差分布不随时间变化，那么这个模型就是平稳的。

# 第13章

# 生存分析

生存分析（survival analysis）是一种描述事物持续时间的方式。生存分析常用于研究人类寿命，但也适用于研究机械和电子元件的“生存期”，或更为通用地研究一个事件将要发生的时间。

如果你认识患有某种绝症的人，那么可能知道什么是“5 年生存率”，即病患从确诊开始 5 年后的存活概率。这个估计值和相关的统计量是生存分析的结果。

本章代码位于 survival.py 中。前言介绍了如何下载和使用本书代码。

## 13.1 生存曲线

生存曲线（survival curve）$S(t)$ 是生存分析中的基本概念。$S(t)$ 是一个函数，将一个持续时间 $t$ 映射到存活时间超过 $t$ 的概率。如果已知持续时间（或“生存期”）的分布，那么就很容易计算出生存曲线，即 CDF 的补函数。

$$S(t) = 1 - \mathrm{CDF}(t)$$

其中 CDF($t$) 是生存期小于或等于 $t$ 的概率。

例如，在全国家庭增长调查数据集中有 11 189 个完整妊娠持续时间的数据。我们可以读取这些数据，计算 CDF。

```
preg = nsfg.ReadFemPreg()
complete = preg.query('outcome in [1, 3, 4]').prglngth
cdf = thinkstats2.Cdf(complete, label='cdf')
```

妊娠结果代码 1、3 和 4 分别代表成功生产、死胎和流产。这个分析排除了引产、宫外孕，以及调查进行时参与者妊娠尚未结束的情况。

DataFrame 的 `query` 方法以一个布尔表达式为参数，为每一行数据计算这个表达式的值，选择结果为 True 的行。

图 13-1（上）展示了妊娠持续时间的 CDF 及其补函数，即生存函数。为了表示生存函数，此处定义了一个对象，对 Cdf 进行封装，并提供适当的接口。

```
class SurvivalFunction(object):
    def __init__(self, cdf, label=''):
        self.cdf = cdf
        self.label = label or cdf.label

    @property
    def ts(self):
        return self.cdf.xs

    @property
    def ss(self):
        return 1 - self.cdf.ps
```

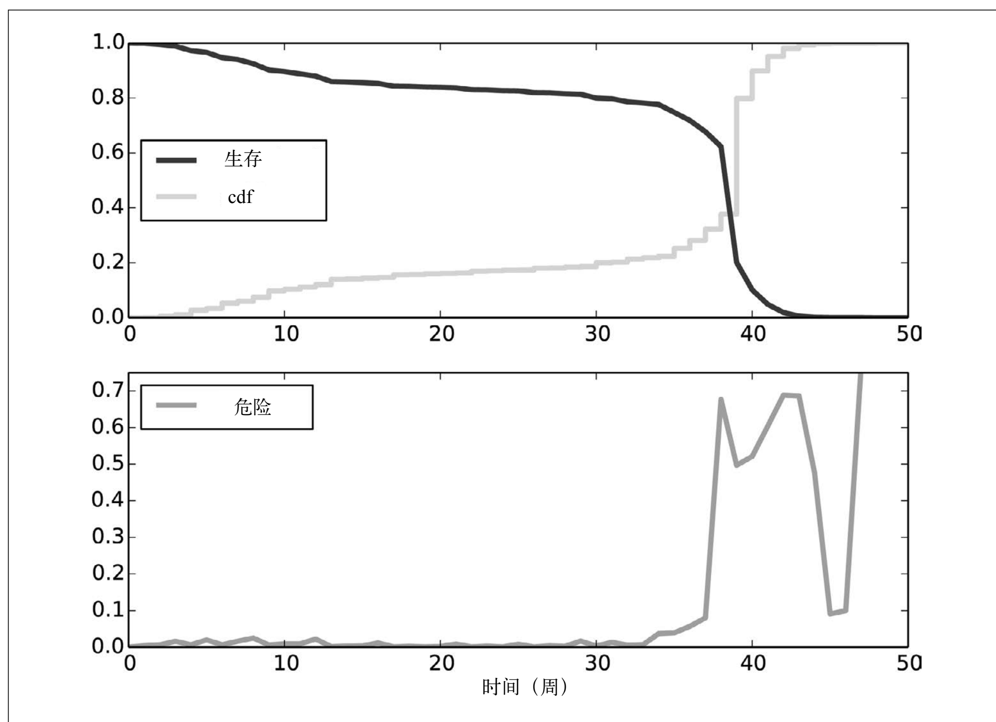



图 13-1：妊娠持续时间的 Cdf 和生存函数（上）及危险函数（下）

SurvivalFunction 有两个属性：ts 是生存期序列，ss 是生存函数。在 Python 中，“属性”是一种可以像变量一样被调用的方法。

SurvivalFunction 可以通过传入生存期的 CDF 进行初始化。

```
sf = SurvivalFunction(cdf)
```

SurvivalFunction 还提供 __getitem__ 和 Prob 方法，用于计算生存函数。

```
# class SurvivalFunction

    def __getitem__(self, t):
        return self.Prob(t)

    def Prob(self, t):
        return 1 - self.cdf.Prob(t)
```

例如，sf[13] 是妊娠持续时间超过 3 个月的比例。

```
>>> sf[13]
0.86022
>>> cdf[13]
0.13978
```

约有 86% 的妊娠持续时间超过 3 个月，约有 14% 未超过 3 个月。

SurvivalFunction 提供了 Render 方法，因此 sf 可以用 thinkplot 中的函数绘制。

```
thinkplot.Plot(sf)
```

图 13-1（上）展示了绘制结果。图中的曲线在 13~26 周几乎是平的，说明很少有妊娠在第三到第六个月期间终止。这条曲线在 39 周附近最为陡峭，39 周正是最常见的妊娠持续时间。

## 13.2　危险函数

从生存函数可以推导出*危险函数*（hazard function）。妊娠持续时间的危险函数从时间 $t$ 映射到在 $t$ 终止的妊娠比例。具体公式如下：

$$\lambda(t) = \frac{S(t) - S(t+1)}{S(t)}$$

公式中的分子是终止于 $t$ 的生存期比例，也就是 PMF($t$)。

SurvivalFunction 提供 MakeHazard 方法，用于计算危险函数。

```
# class SurvivalFunction

    def MakeHazard(self, label=''):
        ss = self.ss
        lams = {}
        for i, t in enumerate(self.ts[:-1]):
            hazard = (ss[i] - ss[i+1]) / ss[i]
            lams[t] = hazard

        return HazardFunction(lams, label=label)
```

`HazardFunction` 对象封装了一个 pandas Series。

```
class HazardFunction(object):

    def __init__(self, d, label=''):
        self.series = pandas.Series(d)
        self.label = label
```

`d` 可以是一个字典对象或其他任何可以初始化 Series 的数据类型，也可以是一个 Series。`label` 是一个字符串，在绘制图形时标识这个 `HazardFunction` 对象。

`HazardFunction` 提供 `__getitem__` 方法，因此可以使用如下代码：

```
>>> hf = sf.MakeHazard()
>>> hf[39]
0.49689
```

如此可见，在所有持续时间达到或超过 39 周的妊娠中，约有 50% 终止于第 39 周。

图 13-1（下）展示了妊娠持续时间的危险函数。在超过 42 周的部分，危险函数只基于少量的数据，因此很不规律。其余部分的曲线形状则符合预期：曲线在 39 周附近最高，前 3 个月比中间 3 个月的值略高。

危险函数不仅自身用处很大，而且还是估计生存曲线的重要工具，下一节将对其进行介绍。

## 13.3 估计生存曲线

如果给出了生存期的 CDF，那么很容易计算生存函数和危险函数。但是，在很多现实世界场景中，我们无法直接度量生存期的分布，必须进行推断。

例如，我们需要跟踪记录一群病患在确诊后的生存时间。不是所有的病患都在同一天确诊，因此在任何时间点，都有一部分病患比其他病患生存时间更长。如果一些病患去世，那么他们的生存时间就是确定的。对于活着的病患，他们的生存时间未知，但存在一个下限。

如果等到所有的病患都去世，那我们就可以计算出生存曲线，但是要评估一项新疗法的效果可不能等那么久！我们需要找到一种方法，使用不完整的信息估计生存曲线。

举个令人高兴一点的例子：我要使用全国家庭增长调查数据，计算调查参与者第一次结婚的年龄。全国家庭增长调查的参与者年龄范围是 14~44 岁，因此这个数据集提供了处于不同生命阶段女性的一个快照。

对于已婚女性，全国家庭增长调查数据集包含了她们的初婚日期以及初婚年龄。对于未婚女性，我们可以得到她们参与调查时的年龄，但无法得知她们会不会结婚，什么时候结婚。

既然已知一些女性的初婚年龄，那么我们可能试图排除未婚女性的数据，只计算已知数据的 CDF。这个想法很糟糕。这么做会产生双重误导的结果：(1) 年龄较大的女性在调查时已婚的可能性更大，因此所占比例偏大；(2) 已婚女性所占比例也偏大！实际上，这样分析得到的结论会是所有女性都已婚，而这显然是不正确的。

## 13.4 Kaplan-Meier估计

在计算初婚年龄时，将未婚女性包括在内不仅是可取的，而且是必要的。这就要用到生存分析中的一个主要算法：Kaplan-Meier 估计（Kaplan-Meier estimation）。

Kaplan-Meier 估计的大致思路是使用数据估计危险函数，然后将危险函数转换为生存函数。要估计初婚年龄的危险函数，我们需要对每个年龄计算：(1) 在这个年龄结婚的女性人数；(2)“有危险的”女性人数，其中包括在这个年龄之前未婚的所有女性。

具体代码如下：

```
def EstimateHazardFunction(complete, ongoing, label=''):

    n = len(complete)
    hist_complete = thinkstats2.Hist(complete)
    sf_complete = SurvivalFunction(thinkstats2.Cdf(complete))

    m = len(ongoing)
    sf_ongoing = SurvivalFunction(thinkstats2.Cdf(ongoing))

    lams = {}
    for t, ended in sorted(hist_complete.Items()):
        at_risk = ended + n * sf_complete[t] + m * sf_ongoing[t]
        lams[t] = ended / at_risk

    return HazardFunction(lams, label=label)
```

`complete` 是完成观测集，即调查参与者结婚的年龄。`ongoing` 是未完成观测集，即未婚女性在参与调查时的年龄。

首先，代码计算女性结婚年龄的直方图 hist_complete，已婚女性的生存函数 sf_complete 以及未婚女性的生存函数 sf_ongoing。

代码中的循环遍历参与者结婚的年龄，每个年龄 t 可以对应到在这个年龄结婚的女性人数 ended，然后计算“有危险”的女性人数，其值为以下值之和：

- ended，即在年龄 t 结婚的参与者人数；
- n * sf_complete[t]，即在年龄 t 之后结婚的参与者人数；
- m * sf_ongoing[t]，即在参与调查时年龄超过 t 的未婚人数，也就是没有在年龄 t 或之前结婚的人数。

危险函数在 t 的估计值为 ended 除以 at_risk 所得的比率。

lams 是一个字典，将 $t$ 映射到 $\lambda(t)$。代码返回的结果为一个 HazardFunction 对象。

## 13.5　婚姻曲线

要测试这个函数，首先需要进行数据清洗和转换。所需的全国家庭增长调查变量如下。

- cmbirth

  参与者的出生日期，所有参与者都有这项数据。

- cimtvw

  参与者的受访日期，所有参与者都有这项数据。

- cmmarrhx

  参与者的初婚日期，如果适用而且已知。

- evrmarry

  如果参与者在受访时已婚则为 1，否则为 0。

前 3 个变量的编码格式为“世纪 – 月”，即距 1899 年 12 月的月数整数值。因此，世纪 – 月值 1 就是 1900 年 1 月。

首先，读取调查参与者文件，替换 cmmarrhx 中的无效值。

```
resp = chap01soln.ReadFemResp()
resp.cmmarrhx.replace([9997, 9998, 9999], np.nan, inplace=True)
```

然后计算每位参与者结婚时的年龄以及受访时的年龄。

```
resp['agemarry'] = (resp.cmmarrhx - resp.cmbirth) / 12.0
resp['age'] = (resp.cmintvw - resp.cmbirth) / 12.0
```

接下来，抽取 complete 数据，即已婚女性的结婚年龄，以及 ongoing，即未婚女性的受访年龄。

```
complete = resp[resp.evrmarry==1].agemarry
ongoing = resp[resp.evrmarry==0].age
```

最后计算危险函数。

```
hf = EstimateHazardFunction(complete, ongoing)
```

图 13-2（上）展示了计算得到的估计危险函数。危险函数值在青少年时期很低，20 多岁时较高，在 30 多岁时呈下降趋势。危险函数在 40 多岁时再次上升，但这是估计过程造成的结果。随着“有危险”的参与者人数下降，很少的结婚女性人数会产生很大的估计危险值。生存函数将会对这种噪音进行平滑处理。

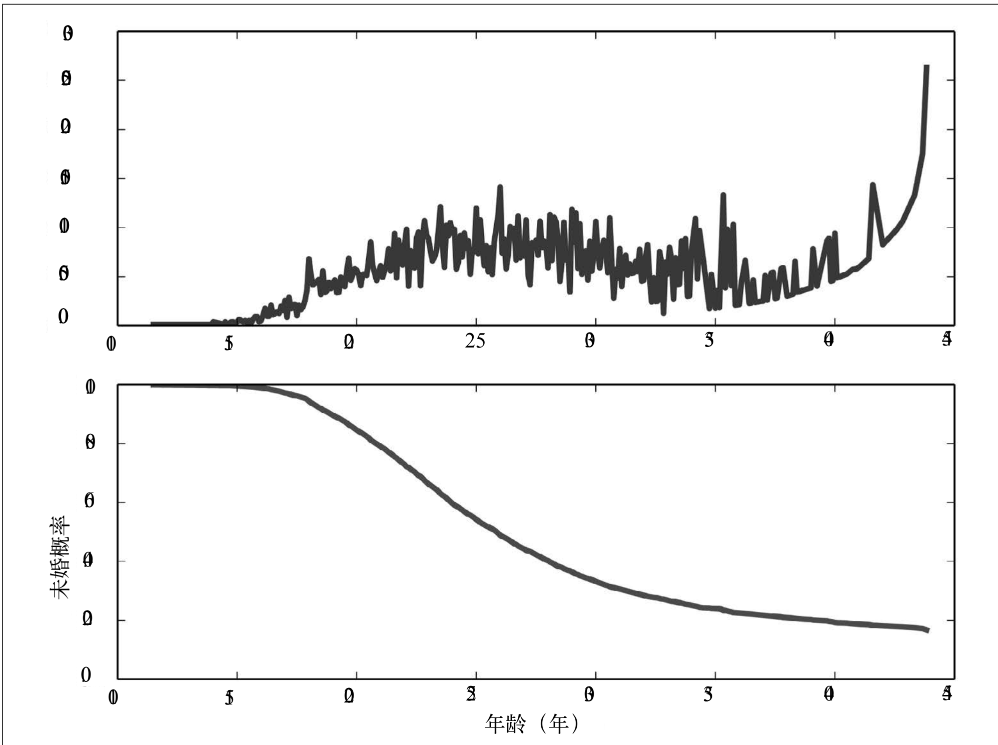



图 13-2：初婚年龄的危险函数（上）和生存函数（下）

## 13.6　估计生存函数

得到危险函数之后，我们就可以估计生存函数了。超过时间 t 仍然生存的概率是在 t 之前的所有时间都生存并且在 t 也生存的概率，即危险补函数的累积乘积：

$$[1-\lambda(0)][1-\lambda(1)]\cdots[1-\lambda(t)]$$

`HazardFunction` 类提供 `MakeSurvival` 方法，计算这个乘积。

```
# class HazardFunction:

    def MakeSurvival(self):
        ts = self.series.index
        ss = (1 - self.series).cumprod()
        cdf = thinkstats2.Cdf(ts, 1-ss)
        sf = SurvivalFunction(cdf)
        return sf
```

`ts` 是估计危险函数的时间序列，`ss` 是危险补函数的累积乘积，也就是生存函数。

受限于 `SurvivalFunction` 的实现方式，我们必须计算 `ss` 的补，生成一个 Cdf，然后才能实例化 `SurvivalFunction` 对象。

图 13-2（下）展示了计算结果。生存曲线在 25~35 岁最为陡峭，这正是大部分女性的结婚年龄。在 35~45 岁，曲线几乎是平直的，说明 35 岁前没有结婚的女性很可能不会结婚。

1986 发表的一篇杂志文章就是基于这样的生存曲线。《新闻周刊》（*Newsweek*）报道称，对于一位 40 岁的未婚女性，其结婚的可能性比“被恐怖分子杀害”的可能性还小。这些统计数据得到广泛报道，成为流行文化的一部分，但那时的数据是错误的（因其基于错误的分析），而且与今天的现实相去更远（由于从那时开始并持续至今的文化变迁）。2006 年，《新闻周刊》发表了另一篇文章，承认了自己的错误。

我建议你了解一下这篇文章的相关背景，其所基于的统计数据以及产生的反响。我们应当引以为戒，提醒自己遵循道德规范，谨慎执行统计分析，在解释分析结果时持有适当的怀疑态度，并将结果准确诚实地呈现给公众。

## 13.7　置信区间

Kaplan-Meier 分析生成生存曲线的一个估计，但量化这个估计的不确定性也很重要。此处的误差来源同样有 3 种：测量误差、抽样误差和建模误差。

在初婚年龄分析中，测量误差可能很小。人们通常都知道自己何时出生、是否已婚，以及结婚时间，而且应该会准确提供这些信息。

我们可以使用重抽样来量化抽样误差。代码如下：

```
def ResampleSurvival(resp, iters=101):
    low, high = resp.agemarry.min(), resp.agemarry.max()
    ts = np.arange(low, high, 1/12.0)
```

```
    ss_seq = []
    for i in range(iters):
        sample = thinkstats2.ResampleRowsWeighted(resp)
        hf, sf = EstimateSurvival(sample)
        ss_seq.append(sf.Probs(ts))

    low, high = thinkstats2.PercentileRows(ss_seq, [5, 95])
    thinkplot.FillBetween(ts, low, high)
```

ResampleSurvival 的参数 resp 是包含参与者信息的 DataFrame，iters 是重抽样的次数。代码计算出年龄序列 ts，用于估算生存函数。

在循环代码内，ResampleSurvival 进行如下操作：

- 使用 ResampleRowsWeighted 对参与者重抽样，10.7 节介绍过 ResampleRowsWeighted；
- 调用 EstimateSurvival，这一函数使用之前章节介绍的过程估算危险曲线和生存曲线；
- 上述函数还能计算 ts 中每个年龄的生存曲线。

ss_seq 是估算得到的生存曲线序列。PercentileRows 使用这一序列，计算第 5 和第 95 百分位秩，返回生存曲线的 90% 置信区间。

图 13-3 展示了 ResampleSurvival 的运行结果，以及前一节中估算的生存函数。与估计曲线不同，这一结果中的置信区间考虑了取样权重。两条曲线的区别表明取样权重对估计有显著的影响——这一点需要牢记。

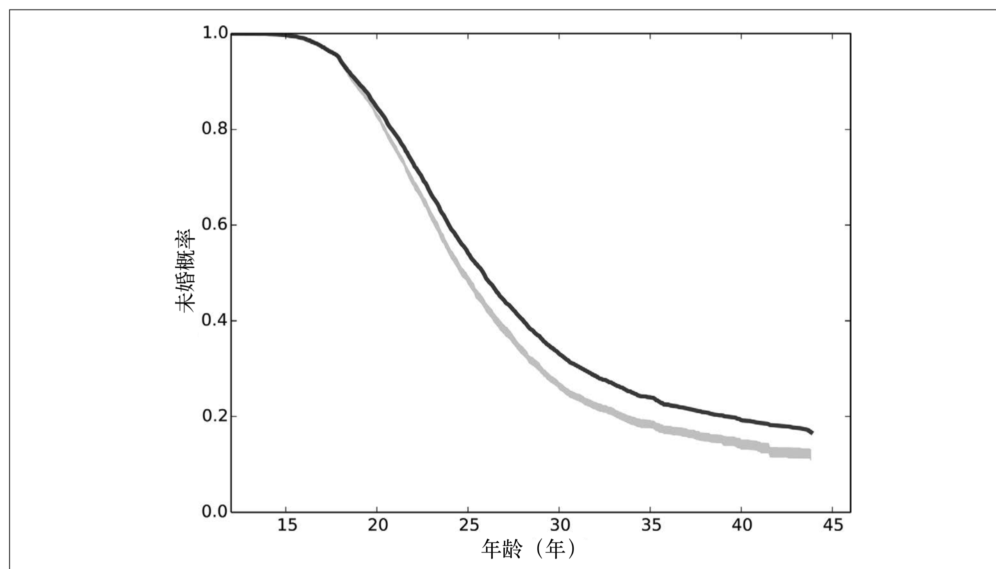



图 13-3：初婚年龄的生存函数以及基于加权重抽样的 90% 置信区间

## 13.8 群组效应

生存分析的一个挑战是，估计曲线的不同部分基于不同的调查参与者群组。曲线在时间 `t` 的部分基于的参与者在受访时年龄不小于 `t`。因此，曲线最左侧的部分包含了所有参与者的数据，而最右侧只包含了年龄最大的那部分参与者。

如果调查参与者的相关特征不随时间变化，那就没有问题。但是在初婚年龄这一问题上，出生于不同时代的女性似乎具有不同的婚姻模式。我们可以将调查参与者按照出生的年代进行分组，来研究这一效应。像这样由出生日期或类似事件定义的组，称为*群组*（cohort），各组之间的差异则称为*群组效应*（cohort effect）。

为了研究全国家庭增长调查婚姻数据中的群组效应，我收集了本书中多处使用的来自 2002 年的第 6 周期数据，9.11 节使用数据来自 2006~2010 的第 7 周期，以及 1995 年第 5 周期。这些数据集一共包含 30 769 位调查参与者。

```
resp5 = ReadFemResp1995()
resp6 = ReadFemResp2002()
resp7 = ReadFemResp2010()
resps = [resp5, resp6, resp7]
```

对 `resp` 中的每个 DataFrame，使用 `cmbirth` 计算每位参与者的出生年代。

```
month0 = pandas.to_datetime('1899-12-15')
dates = [month0 + pandas.DateOffset(months=cm)
         for cm in resp.cmbirth]
resp['decade'] = (pandas.DatetimeIndex(dates).year - 1900) // 10
```

编码变量 `cmbirth` 的值是距离 1899 年 12 月的月数整数值。`month0` 将这个起始日期表示为 Timestamp 对象。对每个出生日期，代码初始化一个包含世纪 – 月值的 `DateOffset`，与 `month0` 相加，得到一个 Timestamp 序列，将其转换为 `DateTimeIndex`，最后从中抽取 `year` 值，计算年代。

为了考虑取样权重，也为了展示由取样误差导致的数据变化，要对数据进行重抽样，将参与者按年代分组，并绘制生存曲线。

```
for i in range(iters):
    samples = [thinkstats2.ResampleRowsWeighted(resp)
               for resp in resps]
    sample = pandas.concat(samples, ignore_index=True)
    groups = sample.groupby('decade')

    EstimateSurvivalByDecade(groups, alpha=0.2)
```

来自全国家庭增长调查的 3 个周期数据使用了不同的取样权重，因此我对每组数据分别进行重抽样，然后用 `concat` 将其合并成一个 DataFrame。参数 `ignore_index` 告诉 `concat` 方法不要按索引匹配参与者，而是创建一个从 0 到 30 768 的新索引。

`EstimateSurvivalByDecade` 绘制每个群组的生存曲线。

```
def EstimateSurvivalByDecade(resp):
    for name, group in groups:
        hf, sf = EstimateSurvival(group)
        thinkplot.Plot(sf)
```

图 13-4 展示了绘制结果，从中可见几个模式。

- 生于 20 世纪 50 年代的女性结婚最早，随后的群组结婚时间越来越晚，至少到 30 岁左右都是如此。
- 生于 20 世纪 60 年代的女性的婚姻模式令人惊讶。在 25 岁之前，她们比上一代人结婚率低。而在 25 岁之后，她们的结婚率上升较快，到 32 岁时已经超过了 50 年代的群组。她们在 44 岁时结婚的可能性比别的群组大得多。

  生于 20 世纪 60 年代的女性，在 1985~1995 年间年龄达到 25 岁。回想起前面提到的 1986 年出版的《新闻周刊》文章，我们可能会认为这篇文章引发了一场结婚潮。这个解释未免有点取巧，但这篇文章及其社会反响的确可能反映了影响这一群组行为的情绪。
- 20 世纪 70 年代群组的模式类似 60 年代。在 25 岁之前，她们比上一代人结婚率低，而到 35 岁则超过了前两个群组。
- 生于 20 世纪 80 年代的女性在 25 岁之前结婚的人更少。80 年代之后的情况就不清楚了，要获得更多数据，我们得等到全国家庭增长调查的下一个周期。

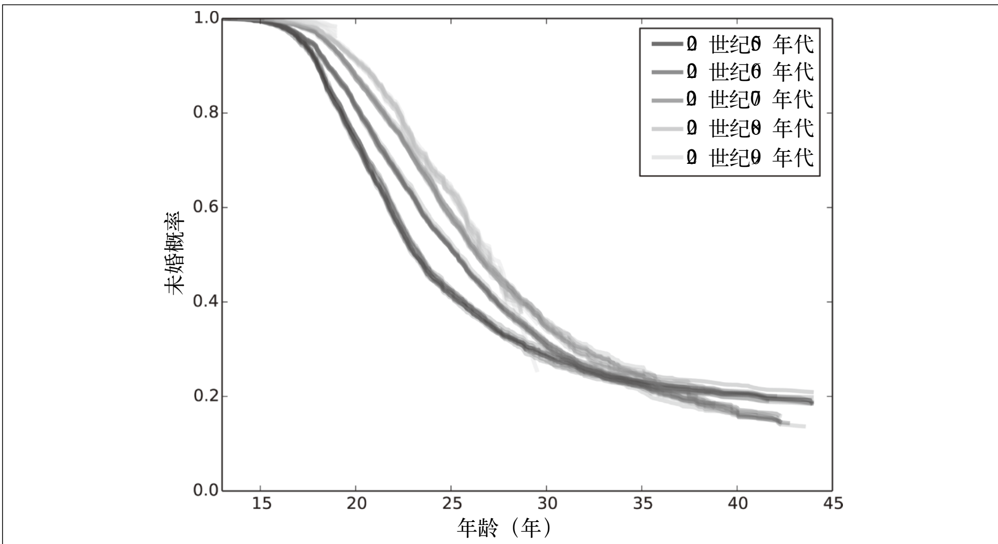



图 13-4：出生于不同年代的调查参与者的生存函数

与此同时，我们还可以进行一些预测。

## 13.9　外推

70 年代群组的生存曲线止于 38 岁左右，80 年代群组则止于 28 岁，而 90 年代群组几乎就没有什么数据。

我们可以从前一个群组“借”些数据，对这些曲线进行外推。HazardFunction 提供一个 `Extend` 方法，可以从另一个较长的 HazardFunction 中复制尾部数据。

```
# class HazardFunction

    def Extend(self, other):
        last = self.series.index[-1]
        more = other.series[other.series.index > last]
        self.series = pandas.concat([self.series, more])
```

13.2 节介绍过，HazardFunction 包含一个将 $t$ 映射到 $\lambda(t)$ 的 Series。`Extend` 方法找到 `self.series` 中的最后一个索引值 `last`，选择 `other` 中位于 `last` 之后的值，并将这些值附加在 `self.series` 上。

现在我们可以使用前一个年代群组的数据来扩展每个群组的 HazardFunction。

```
def PlotPredictionsByDecade(groups):
    hfs = []
    for name, group in groups:
        hf, sf = EstimateSurvival(group)
        hfs.append(hf)

    thinkplot.PrePlot(len(hfs))
    for i, hf in enumerate(hfs):
        if i > 0:
            hf.Extend(hfs[i-1])
        sf = hf.MakeSurvival()
        thinkplot.Plot(sf)
```

`groups` 是一个 GroupBy 对象，其中的参与者按出生年代进行了分组。第一个循环计算了每个组的 HazardFunction。

第二个循环使用前一组的数据对每个组的 HazardFunction 进行扩展，而前一组的数据可能包含来自更前一组的数据，依此类推。然后代码将每个 HazardFunction 转换为一个 SurvivalFunction，并绘制其图形。

图 13-5 展示了代码运行的结果。图中去除了 50 年代的群组，使预测更清晰可见。这些结果表明，到 40 岁时，最近的群组会与 60 年代群组重合，只剩不到 20% 的人还未结婚。

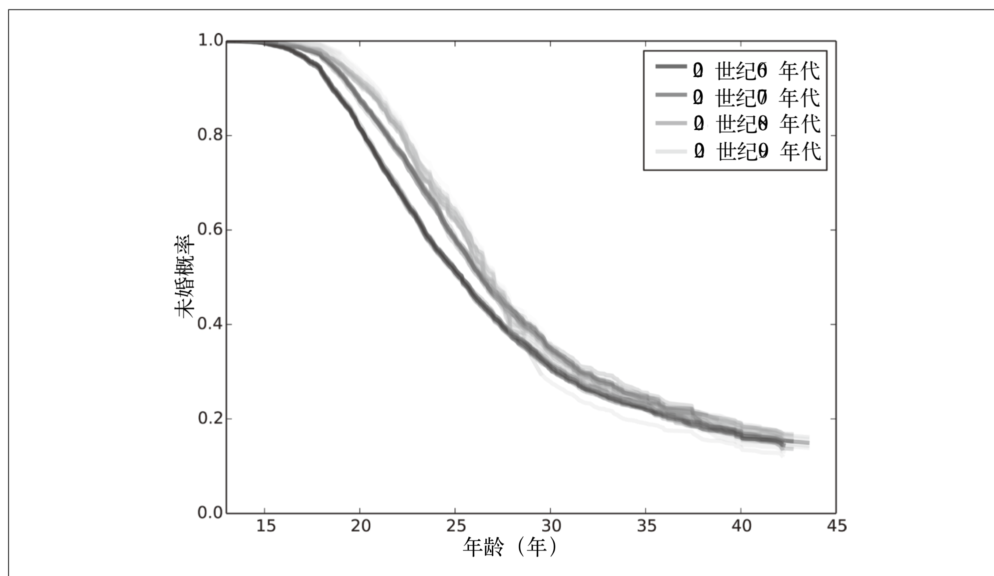



图 13-5：生于不同年代的参与者的生存曲线，较晚的群组带有预测值

## 13.10　预期剩余生存期

给定一个生存曲线，我们可以计算出以当前年龄为参数的预期剩余生存期函数。例如，给定 13.1 节中的妊娠持续时间的生存函数，我们可以计算出距离婴儿出生的预期时间。

第一步是得到生存期的 PMF。`SurvivalFunction` 为此提供了一个方法。

```
# class SurvivalFunction

    def MakePmf(self, filler=None):
        pmf = thinkstats2.Pmf()
        for val, prob in self.cdf.Items():
            pmf.Set(val, prob)

        cutoff = self.cdf.ps[-1]
        if filler is not None:
            pmf[filler] = 1-cutoff

        return pmf
```

请记住，SuvivalFunction 包含生存期 Cdf。代码中的循环将 Cdf 中的值和概率复制到一个 Pmf 中。

`cutoff` 是 Cdf 中最大的概率值，如果 Cdf 包含全部数据，则此值为 1，否则此值小于 1。如果 Cdf 的数据不完整，代码将插入一个给定的值 `filler`，从而补全数据。

妊娠持续时间的 Cdf 是完整的，因此不需要考虑这个细节。

下一步是计算预期剩余生存期，其中的“预期”是平均的意思。SurvivalFunction 也为此提供了一个方法。

```
# class SurvivalFunction

    def RemainingLifetime(self, filler=None, func=thinkstats2.Pmf.Mean):
        pmf = self.MakePmf(filler=filler)
        d = {}
        for t in sorted(pmf.Values())[:-1]:
            pmf[t] = 0
            pmf.Normalize()
            d[t] = func(pmf) - t

        return pandas.Series(d)
```

RemainingLifetime 的参数 filler 会传递给 MakePmf，参数 func 是用于总结剩余生命期分布的函数。

pmf 是从 SurvivalFunction 得到的生存期 Pmf。d 是包含计算结果的字典，即从当前年龄 t 到预期剩余生命期的映射。

代码中的循环遍历 Pmf 中的值。对每个值，计算在生命期超过 t 的条件下，生命期的条件分布。具体做法是从 Pmf 中一次移除一个值，对剩余值重新进行正态化。

随后，代码使用 func 总结得到的条件概率。这个示例中的结果是，在妊娠持续时间超过 t 的条件下，妊娠持续时间的均值。减去 t 则得到了剩余妊娠持续时间的均值。

图 13-6（左）展示了以当前妊娠持续时间为参数的预期持续妊娠时间函数。例如，在第 0 周，预期剩余持续时间约为 34 周。这个值比足月妊娠（39 周）小，因为前 3 个月中的终止妊娠将均值拉低了。

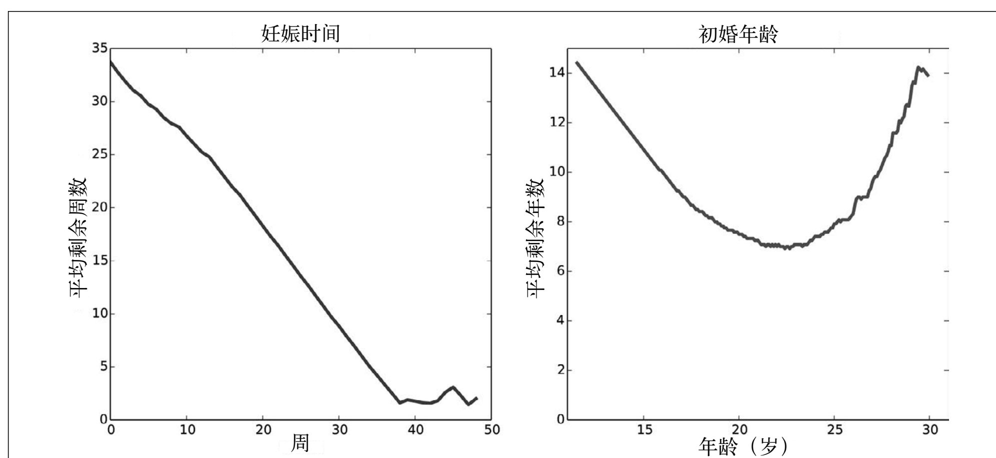



图 13-6：妊娠持续时间的预期剩余生存期（左）和初婚剩余年数（右）

图中的曲线在前 3 个月缓慢下降。到第 13 周，预期剩余生存期只下降了 9 周，达到了 25 周。之后曲线下降较快，每过 1 周约下降 1 周。

在第 37~42 周，曲线走平，值在 1~2 周。在这一时间段的任何时刻，预期剩余生存期都是相同的，随着每一周的过去，目标却没有更近。具有这种属性的过程称为*无记忆的*（memoryless），因为过去的时间对预测不产生任何影响。这种行为就是产科护士们恼人的口头禅“随时可能发动”（any day now）的数学基础。

图 13-6（右）展示了到初婚年龄的剩余时间中位数函数，参数为年龄。对于一个 11 岁的女孩，到初婚年龄的中位数时间约为 14 年。这条曲线持续下降，到 22 岁附近剩余时间中位数保持在 7 年左右，之后开始上升，到 30 岁又回到了最开始的剩余时间：14 年。

根据这一数据，年轻女性的剩余“生存期”呈下降趋势。具有这一属性的机械组件称为 NBUE，即“新比旧好”（new better than used in expectation），意思是新部件持续时间更长。

年龄超过 22 岁的女性到初婚的剩余时间呈上升趋势。具有这种属性的组件称为 UBNE，即“旧比新好”（used better than new in expectation）。也就是说，部件越旧，预期持续的时间越长。新生儿和癌症病患也属于 UBNE，他们的生存时间越长，预期寿命就越长。

对于初婚剩余时间，我计算的是中位数而非均值，因为 Cdf 不是完整的。生存曲线表明，约有 20% 的参与者不会在 44 岁之前结婚。这些女性的初婚年龄属于未知，有可能并不存在，因此无法计算均值。

我处理这些未知数据的方法，是将其替换为一个代表无穷大的特殊值 `np.inf`。这样可以使所有年龄的均值为无穷大，而只要超过 50% 的剩余生存期为有限值，中位数就是有意义的。30 岁之前的数据都满足这一条件，而之后就很难定义一个有意义的预期剩余生存期了。

计算和绘制这些函数的代码如下：

```
rem_life1 = sf1.RemainingLifetime()
thinkplot.Plot(rem_life1)

func = lambda pmf: pmf.Percentile(50)
rem_life2 = sf2.RemainingLifetime(filler=np.inf, func=func)
thinkplot.Plot(rem_life2)
```

`sf1` 是妊娠持续时间的生存函数，我们可以使用默认参数调用 `RemainingLifetime`。

`sf2` 是初婚年龄的生存函数，`func` 函数以 Pmf 为参数，计算其中位数（第 50 百分位秩）。

## 13.11　练习

本章练习的参考答案位于 chap13soln.py 中。

- 练习 13.1

在全国家庭增长调查的第 6 和第 7 周期中，变量 `cmdivorcx` 是调查参与者第一次婚姻的离婚日期（如果存在的话），编码为世纪 - 月。

请计算以离婚为终止的婚姻持续时间，以及到目前为止仍然延续的婚姻的持续时间。请估计婚姻持续时间的危险函数和生存函数。

请使用重抽样，处理取样权重的影响，并绘制几个重抽样的结果数据，可视化展示取样误差。

请思考如何将调查参与者按出生年代进行分组，以及如何按初婚年龄进行分组。

## 13.12 术语

- 生存分析（survival analysis）
  描述和预测生存期（或直到某事件发生的时间）的一组方法。
- 生存曲线（survival curve）
  一个函数，将时间 `t` 映射到在 `t` 之后仍然存活的概率。
- 危险函数（hazard function）
  一个函数，将时间 `t` 映射到在 `t` 之前存活的人中在 `t` 时刻死亡的比例。
- Kaplan-Meier 估计（Kaplan-Meier estimation）
  估计危险函数和生存函数的一种算法。
- 群组（cohort）
  在特定的事件间隔内，由某个事件（如出生日期）决定的一组对象。
- 群组效应（cohort effect）
  群组之间的差异。
- NBUE
  预期剩余生存期的一个属性，“新比旧好”（New better than used in expectation）。
- UBNE
  预期剩余生存期的一个属性，“旧比新好”（Used better than new in expectation）。

第 14 章

# 分析方法

本书主要讨论的是计算方法，如模拟和重抽样，但书中解决的一些问题还有更高效的分析方法可用。

本章将介绍一些分析方法，解释其工作原理，末尾还将给出一些建议，告知如何将探索性数据分析的计算和分析方法进行结合。

本书代码位于 normal.py 中。前言介绍了如何下载和使用本书代码。

## 14.1 正态分布

让我们回顾一下 8.3 节的问题：

> 假设你是一位科学家，在自然保护区中研究大猩猩。对 9 只大猩猩称重得到的样本均值 $\bar{x} = 90$kg，样本标准差 $S = 7.5$kg。如果使用 $\bar{x}$ 估计总体均值，那么这一估计值的标准误差是多少？

回答这个问题需要计算 $\bar{x}$ 的抽样分布。在 8.3 节，我们通过对实验（称重 9 只大猩猩）进行模拟，计算每次模拟实验的 $\bar{x}$，收集估计值的分布，然后得到近似的分布。

这一近似过程的结果是抽样分布。然后，我们使用这个抽样分布，计算出标准误差和置信区间。

- 抽样分布的标准差是估计值的标准误差。示例中的抽样分布标准差约为 2.5kg。

- 抽样分布的第 5 和第 95 百分位秩的间隔为 90% 置信区间。如果多次运行实验，估计值有 90% 的概率位于这个间隔内。示例中的 90% 置信区间为 (86, 94)kg。

现在我们要用统计方法进行同样的计算。已知成年雌性大猩猩的体重大致符合正态分布，因此我们可以利用这一条件。正态分布有两个特性，使其特别适合于分析：正态分布在线性变换和加法下是“封闭的”（closed）。为了解释这句话的意思，我们需要用到一些符号。

如果一个数值 $X$，符合参数为 $\mu$ 和 $\sigma$ 的正态分布，则可以记为：

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

其中符号 ~ 表示“符合分布”，草体字母 $\mathcal{N}$ 代表“正态”。

$X$ 的一个线性变换形为 $X'=aX+b$，其中 $a$ 和 $b$ 为具体数字。如果 $X'$ 与 $X$ 属于同一个分布族，那么这个分布族就是封闭的。正态分布具有这一属性。如果 $X \sim \mathcal{N}(\mu, \sigma^2)$，那么

$$X' \sim \mathcal{N}(a\mu+b, a^2\sigma^2) \quad (1)$$

正态分布在加法下也是封闭的。如果 $Z=X+Y$，并且 $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$，$Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$，那么

$$Z \sim \mathcal{N}(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2) \quad (2)$$

对于 $Z=X+X$ 的特殊情况，有

$$Z \sim \mathcal{N}(2\mu_X, 2\sigma_X^2)$$

通常，如果从 $X$ 中抽取 $n$ 个值求和，结果将有：

$$Z \sim \mathcal{N}(n\mu_X, n\sigma_X^2) \quad (3)$$

## 14.2 抽样分布

万事俱备，我们现在可以开始计算 $\bar{x}$ 的抽样分布了。请记住，$\bar{x}$ 的计算方法是：对 $n$ 只大猩猩称重，得到体重总和，然后除以 $n$。

假设大猩猩体重 $X$ 大致符合正态分布：

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

如果测量 $n$ 只大猩猩的体重，那么由公式 (3) 可知，体重之和 $Y$ 的分布为：

$$Y \sim \mathcal{N}(n\mu, n\sigma^2)$$

将公式 (1) 中的 $a$ 替换为 $1/n$，即得到 $Y$ 除以 $n$ 所得的样本均值 $Z$ 的分布：

$$Z \sim \mathcal{N}(\mu, \sigma^2/n)$$

$Z$ 的分布就是 $\bar{x}$ 的抽样分布。$Z$ 的均值为 $\mu$，说明 $\bar{x}$ 是 $\mu$ 的无偏估计。抽样分布的方差为 $\sigma^2/n$。

因此，抽样分布的标准差，即估计值的标准误差，为 $\sigma/\sqrt{n}$。在大猩猩体重示例中，$\sigma$ 为 7.5kg，$n$ 为 9，因此标准误差为 2.5kg。这与使用模拟方法得到结果完全一致，但计算过程要快得多！

我们也可以使用抽样分布计算置信区间。$\bar{x}$ 的 90% 置信区间是 $Z$ 的第 5 和第 95 百分位秩的间隔。$Z$ 符合正态分布，因此可以通过 CDF 反函数计算出百分位秩。

正态分布的 CDF 或其反函数没有闭式解，但有计算速度很快的数值方法，这些方法位于 SciPy 软件包中（参见 5.2 节）。`thinkstats2` 提供一个封装函数，使 SciPy 函数更易使用。

```
def EvalNormalCdfInverse(p, mu=0, sigma=1):
    return scipy.stats.norm.ppf(p, loc=mu, scale=sigma)
```

对于给定的概率 `p`，`EvalNormalCdfInverse` 返回参数为 `mu` 和 `sigma` 的正态分布的相应百分位秩。要得到 $\bar{x}$ 的 90% 置信区间，需要计算第 5 和第 95 百分位秩：

```
>>> thinkstats2.EvalNormalCdfInverse(0.05, mu=90, sigma=2.5)
85.888

>>> thinkstats2.EvalNormalCdfInverse(0.95, mu=90, sigma=2.5)
94.112
```

结果表明，如果多次运行实验，估计值 $\bar{x}$ 位于区间 (85.9, 94.1) 内的概率为 90%。这依然与使用模拟方法得到的结果一致。

## 14.3　表示正态分布

为了使这些计算更加容易，我定义了一个 `Normal` 类，用于表示正态分布，并对之前章节定义的公式进行编码。具体定义如下：

```
class Normal(object):

    def __init__(self, mu, sigma2):
        self.mu = mu
```

```
        self.sigma2 = sigma2

    def __str__(self):
        return 'N(%g, %g)' % (self.mu, self.sigma2)
```

下面的代码可以实例化一个表示大猩猩体重分布的 Normal 对象。

```
>>> dist = Normal(90, 7.5**2)
>>> dist
N(90, 56.25)
```

Normal 提供 Sum 方法，参数为样本大小 n，使用公式 (3) 计算并返回 n 个值之和的分布。

```
def Sum(self, n):
    return Normal(n * self.mu, n * self.sigma2)
```

Normal 类也可以使用公式 (1) 进行乘法和除法运算。

```
def __mul__(self, factor):
    return Normal(factor * self.mu, factor**2 * self.sigma2)

def __div__(self, divisor):
    return 1 / divisor * self
```

因此，我们可以使用下面的代码，计算样本规模为 9 的均值的抽样分布。

```
>>> dist_xbar = dist.Sum(9) / 9
>>> dist_xbar.sigma
2.5
```

和前一节的结果一样，抽样分布的标准差为 2.5kg。最后，Normal 还提供 Percentile 方法，计算置信区间。

```
>>> dist_xbar.Percentile(5), dist_xbar.Percentile(95)
85.888 94.113
```

这和之前的结果一样。本章稍后还将用到 Normal 类，但在进行具体编码之前，我们还需要了解更多的分析知识。

## 14.4 中心极限定理

前面介绍过，将正态分布中抽取的值相加求和，得到的结果符合正态分布。大多数其他的分布类型不具有这种属性。如果将从其他分布中抽取的值相加，结果通常并不符合分析分布。

但是，如果从几乎任意分布中抽取 n 个值求和，随着 n 的增加，总和的分布会逐渐近似正态分布。

具体地说，如果值的分布具有均值 $\mu$，标准差 $\sigma$，那么其中 $n$ 个值的总和大致符合 $\mathcal{N}(n\mu, n\sigma^2)$。

这个结果称为*中心极限定理*（Central Limit Theorem，CLT）。中心极限定理是进行统计分析最有效的工具之一，但使用时必须注意以下几点：

- 必须独立抽取分布中的值，如果这些值是相关的，那么 CLT 就不适用（这在实际应用中很少遇到）；
- 值必须来自同一个分布（但是这个条件可以放宽）；
- 取值分布的均值和方差必须是有限值，因此很多 Pareto 分布都不能使用 CLT；
- 结果分布的收敛速度取决于分布的偏度，抽取自指数分布的值总和在 `n` 很小时就可以收敛，抽取自对数正态分布的值总和收敛则需要较大的 `n` 值。

中心极限定理解释了为何自然界中存在如此多的正态分布。生命体的很多特征都受基因和环境因素的影响，这些效果是累加的。我们测量到的特征是大量微小效应的总和，因此结果趋于正态分布。

## 14.5 检验CLT

让我们来进行一些实验，看看中心极限定理如何使用，何时不能使用。首先尝试指数分布。

```
def MakeExpoSamples(beta=2.0, iters=1000):
    samples = []
    for n in [1, 10, 100]:
        sample = [np.sum(np.random.exponential(beta, n))
                  for _ in range(iters)]
        samples.append((n, sample))
    return samples
```

`MakeExpoSamples` 生成指数值总和的一个样本（这里的“指数值”指“从指数分布中抽取的值”）。`beta` 是这个指数分布的参数，`iters` 是待生成的总和数量。

这个函数需要从内向外进行解读。代码每次调用 `np.random.exponential` 都会生成一个包含 `n` 个指数值的序列，然后求和。`sample` 是这些总和的列表，长度为 `iters`。

`n` 和 `iters` 很容易混淆：`n` 是每个总和中的值的数量；`iters` 是为了获得这些和值分布而求和的次数。

`MakeExpoSamples` 返回 `(n, sample)` 对的一个列表，下面的代码绘制其中的正态概率图。

```
def NormalPlotSamples(samples, plot=1, ylabel=''):
    for n, sample in samples:
        thinkplot.SubPlot(plot)
```

```
        thinkstats2.NormalProbabilityPlot(sample)

        thinkplot.Config(title='n=%d' % n, ylabel=ylabel)
        plot += 1
```

NormalPlotSamples 以 MakeExpoSamples 返回的列表为参数，生成一行正态概率图。

图 14-1（第一行）展示了 NormalPlotSamples 生成的结果。当 n=1 时，总和仍然符合指数分布，因此绘制出的正态概率图不是一条直线。但当 n=10 时，总和大致符合正态分布。当 n=100 时，结果已经与正态分布没有区别了。

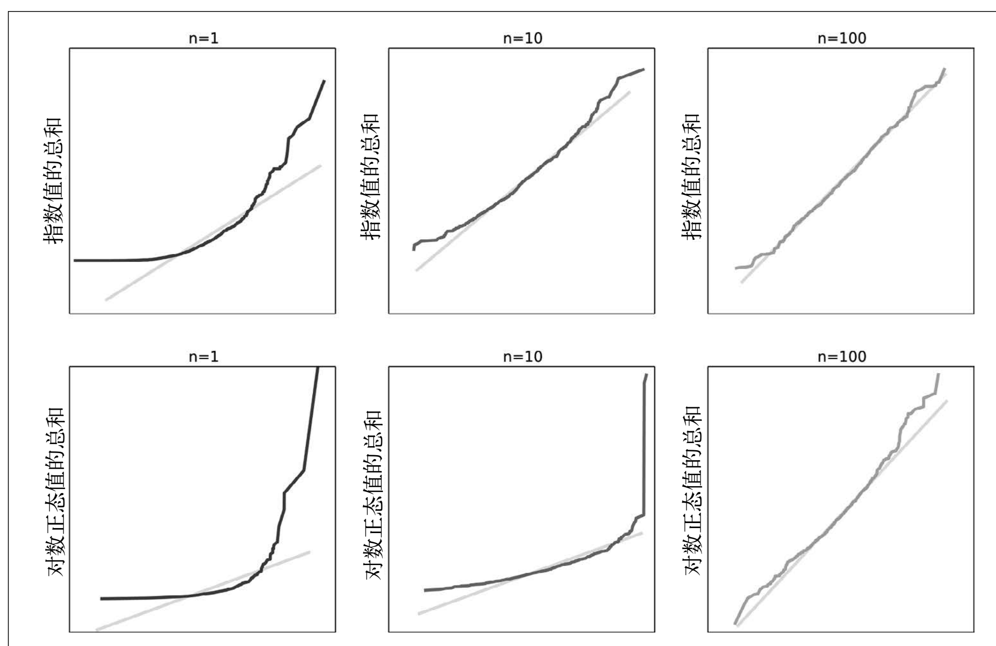



图 14-1：指数值的总和分布（第一行）以及对数正态值的总和分布（第二行）

图 14-1（第二行）展示了对数正态分布产生的类似结果。对数正态分布通常比指数分布的偏度大，因此总和分布的收敛速度较慢。当 n=10 时，图中的正态概率图完全不像直线，但当 n=100 时结果就大致符合正态分布了。

Pareto 分布比对数正态分布偏度更大。很多 Pareto 分布的均值和方差不是有限值，因此不能使用中心极限定理。图 14-2（第一行）展示了 Pareto 值的总和分布。即使当 n=100 时，得到的正态概率图也远非直线。

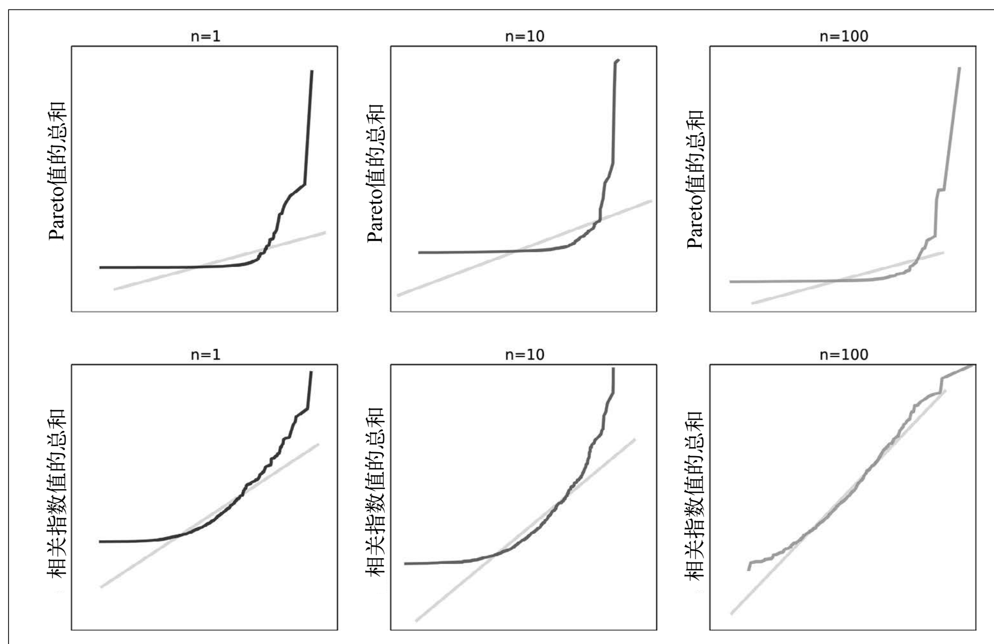



图 14-2：Pareto 值的总和分布（第一行）以及相关指数值的总和分布（第二行）

前面还提到，如果抽取的值是相关的，CLT 也不适用。对此，可以从指数分布生成相关值进行检验。生成相关值的方法为：(1) 生成相关的正态值；(2) 适用正态 CDF 将这些值转换为均匀分布；(3) 使用指数 CDF 反函数，将均匀值转换为指数值。

`GenerateCorrelated` 使用序列相关 `rho`，生成包含 `n` 个正态值的迭代器。

```
def GenerateCorrelated(rho, n):
    x = random.gauss(0, 1)
    yield x

    sigma = math.sqrt(1 - rho**2)
    for _ in range(n-1):
        x = random.gauss(x*rho, sigma)
        yield x
```

结果序列的第一个值是标准正态值，其后的每个值都依赖前一个值。如果前一个值为 `x`，那么下一个值的均值为 `x*rho`，方差为 `1-rho**2`。请注意，`random.gauss` 的第二个参数是标准差，而不是方差。

`GenerateExpoCorrelated` 以 `GenerateCorrelated` 产生的序列为参数，将其转换为指数分布。

```
def GenerateExpoCorrelated(rho, n):
    normal = list(GenerateCorrelated(rho, n))
    uniform = scipy.stats.norm.cdf(normal)
```

```
        expo = scipy.stats.expon.ppf(uniform)
        return expo
```

normal 是相关正态值的一个列表。uniform 是 0~1 均匀分布的值序列。expo 是指数值的相关序列。ppf 代表“百分点函数”（percent point function），即 CDF 反函数。

图 14-2（第二行）展示了 rho=0.9 时，相关指数值的总和分布。取值的相关性使收敛速度变慢，但当 n=100 时，正态概率图也近似直线。由此可见，虽然严格说来，当取值相关时 CLT 不适用，但是在实际应用中，中等程度的相关性影响并不大。

这些实验是为了展示中心极限定理的工作原理，以及中心极限定理的不适用情况。接下来我们要讨论如何应用这一定理。

## 14.6　应用CLT

为了介绍中心极限定理的作用，让我们回顾一下 9.3 节的示例：检验第一胎和其他新生儿的妊娠时间均值的明显差异。这个差异约为 0.078 周。

```
>>> live, firsts, others = first.MakeFrames()
>>> delta = firsts.prglngth.mean() - others.prglngth.mean()
0.078
```

请记住假设检验的逻辑：计算一个 $p$ 值，即在原假设条件下，所观测到差异出现的概率，如果这个值很小，就认为观测到的差异不太可能是偶然产生的。

在这个示例中，原假设为：第一胎和其他新生儿的妊娠时间具有相同的分布。因此，我们可以使用下面的代码，计算均值的抽样分布。

```
dist1 = SamplingDistMean(live.prglngth, len(firsts))
dist2 = SamplingDistMean(live.prglngth, len(others))
```

这两个抽样分布基于相同的总体，即所有成功生产的新生儿。SamplingDistMean 以值序列和样本规模为参数，返回代表抽样分布的 Normal 对象。

```
def SamplingDistMean(data, n):
    mean, var = data.mean(), data.var()
    dist = Normal(mean, var)
    return dist.Sum(n) / n
```

mean 和 var 分别为 data 的均值和方差。我们使用正态分布 dist 对数据进行近似。

在这个示例中，数据不符合正态分布，因此这个近似并不理想。但我们随后计算了 dist.Sum(n) / n，即 n 个值均值的抽样分布。虽然数据不符合正态分布，但根据中心极限定理，均值的抽样分布是符合正态分布的。

接下来，我们计算均值差异的抽样分布。`Normal` 类可以使用公式 (2) 进行两个分布的减法。

```
def __sub__(self, other):
    return Normal(self.mu - other.mu,
                  self.sigma2 + other.sigma2)
```

因此，下面的代码可以算出均值差异的抽样分布。

```
>>> dist = dist1 - dist2
N(0, 0.0032)
```

得到的分布均值为 0。我们认为来自同一分布的两个样本的平均均值相同，因此这个结果非常合理。抽样分布的方差为 0.003 2。

`Normal` 提供 `Prob` 方法，计算正态 CDF。我们可以使用 `Prob` 方法，计算原假设条件下，值为 `delta` 的差异出现的概率。

```
>>> 1 - dist.Prob(delta)
0.084
```

这个结果表明，单侧检验的 $p$ 值为 0.084。我们还可以计算双侧检验的 $p$ 值。

```
>>> dist.Prob(-delta)
0.084
```

结果和单侧检验的 $p$ 值相同，因为正态分布是对称的。尾部的和为 0.168，与 9.3 节的估计值 0.17 相符。

## 14.7　相关检验

9.5 节使用了置换检验，检验新生儿体重与母亲年龄的相关性，认为这一相关是统计显著的，$p$ 值小于 0.001。

现在，我们可以用分析方法完成同样的工作，所用的方法基于这样一个数学结论：对于符合正态分布且互不相关的两个变量，生成大小为 $n$ 的样本，算出 Pearson 相关系性 $r$，然后计算变换相关性：

$$t = r\sqrt{\frac{n-2}{1-r^2}}$$

$t$ 符合参数为 $n$−2 的学生 $t$ 分布（Student's t-distribution）。$t$ 分布是一个分析分布，可以使用伽玛函数快速计算其 CDF。

我们可以利用这一结论，计算原假设条件下，相关性的抽样分布，即：如果从正态分布生成不相关的序列，它们之间相关性的分布如何？`StudentCdf` 函数以样本规模 `n` 为参数，返

回相关性的抽样分布。

```
def StudentCdf(n):
    ts = np.linspace(-3, 3, 101)
    ps = scipy.stats.t.cdf(ts, df=n-2)
    rs = ts / np.sqrt(n - 2 + ts**2)
    return thinkstats2.Cdf(rs, ps)
```

`ts` 是一个 NumPy 数组，其中包含 $t$ 值，即变换相关性。`ps` 包含相应的概率值，由 Scipy 包中的学生 $t$ 分布 CDF 计算得到。$t$ 分布的参数 `df` 代表“自由度”（degree of freedom）。本书对自由度一词不进行解释，你可以参考 http://en.wikipedia.org/wiki/Degrees_of_freedom_(statistics)。

从 `ts` 得到相关系数要进行逆变换。

$$r = t / \sqrt{n - 2 + t^2}$$

得到的结果是原假设条件下 $r$ 的抽样分布。图 14-3 展示了计算得到的分布以及 9.5 节通过重抽样得到的分布，二者相差无几。虽然实际分布并不是正态分布，但 Pearson 相关系数是基于样本均值和方差。根据中心极限定理，即使数据不符合正态分布，这些基于时刻（moment-based）的统计量也是符合正态分布的。

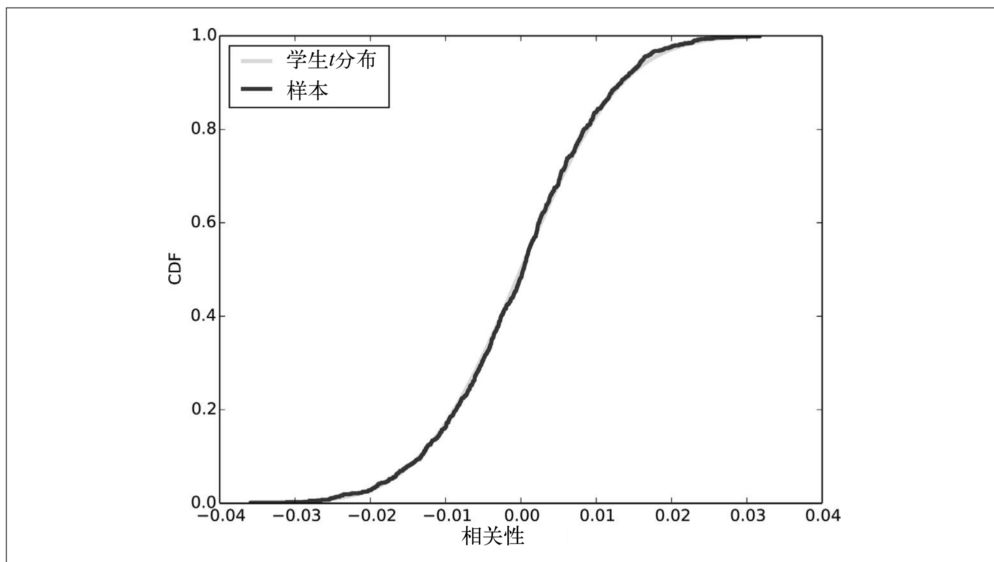



图 14-3：不相关的正态变量之间相关性的抽样分布

从图 14-3 中可以看出，观测相关性为 0.07，如果变量真的没有相关性，这个值出现的可能性很小。使用分析分布，我们可以算出这个可能性有多小。

```
    t = r * math.sqrt((n-2) / (1-r))
    p_value = 1 - scipy.stats.t.cdf(t, df=n-2)
```

先计算对应于 r=0.07 的 t 值，然后计算在 t 处的 $t$ 分布值，结果为 6.4e-12。这个示例展现了分析方法的一个优势：可以计算非常小的 $p$ 值，但实际应用中通常并不会有这样的需求。

## 14.8 卡方检验

9.7 节使用了卡方统计量，检验一个骰子是否有问题。卡方统计量度量实际值与预期值的正态化偏差总和。

$$x^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

卡方统计量用途很广，原因之一是它在原假设条件下的抽样分布是分析分布，最巧的是 1，这个分布就称为卡方分布。和 $t$ 分布一样，卡方 CDF 可以使用伽马函数快速计算。

SciPy 提供了卡方分布的实现，可以用来计算卡方统计量的抽样分布。

```
def ChiSquaredCdf(n):
    xs = np.linspace(0, 25, 101)
    ps = scipy.stats.chi2.cdf(xs, df=n-1)
    return thinkstats2.Cdf(xs, ps)
```

图 14-4 展示了分析方法得到的结果以及通过重抽样得到的分布。二者非常近似，尾部特别贴合，这部分正是人们通常最关心的部分。

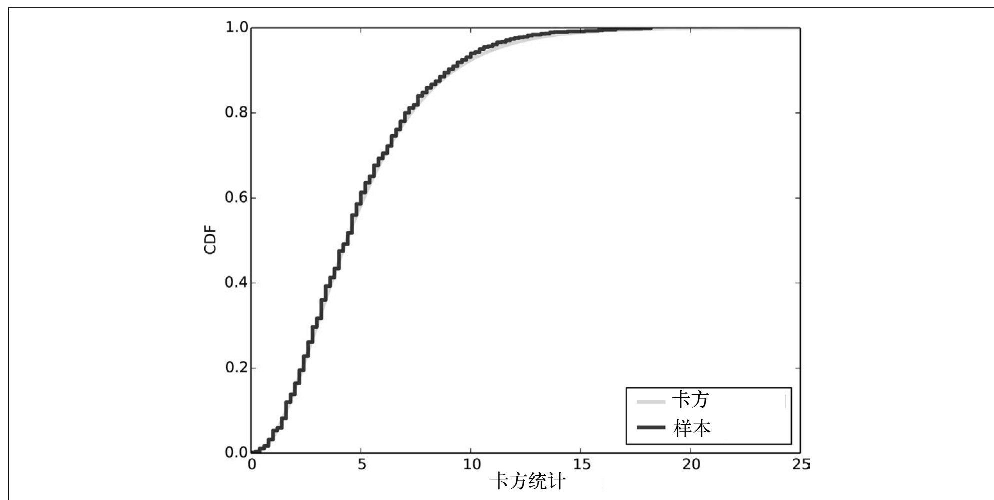



图 14-4：正常六面骰子的卡方统计量抽样分布

注 1：其实并非巧合。

我们可以使用这个分布，计算观测到的检验统计量 chi2 的 $p$ 值。

```
p_value = 1 - scipy.stats.chi2.cdf(chi2, df=n-1)
```

得到的结果为 0.041，与 9.7 节中的结果一致。

卡方分布的参数也是“自由度”。在这里，正确的参数值为 n-1，其中 n 为投掷结果概率表的大小：6。这个参数有时很难选择。说实话，我每次都要等到生成图 14-4 这样的结果，将重抽样的结果与分析结果进行比对后，才能确信自己选对了参数。

## 14.9 讨论

这本书主要讨论计算方法，如重抽样和置换。与分析方法相比，计算方法具有以下几个优势。

- 易于解释和理解。例如，统计学入门课程中最难的一个内容是假设检验。很多学生无法真正理解 $p$ 值是什么。第 9 章中介绍的方法模拟原假设并计算检验统计量，从而可以使基础概念更加清晰。
- 稳健灵活。分析方法经常基于一些现实中未必成立的假设条件。计算方法的假设条件较少，更容易调整和扩展。
- 可以调试。分析方法常常类似一个黑盒子：输入数据就得到结果。但在使用分析方法时，人们很容易犯下难以察觉的错误，很难确信得到的结果是正确的，万一结果错误也很难发现问题。计算方法适合进行增量开发和测试，得到的结果更加可信。

但是计算方法也有一个缺点：有时速度很慢。考虑到这些优缺点，我建议你采取以下过程。

(1) 在探索阶段使用计算方法。如果得到了问题的统计解答，运行速度又还不错，就可以收工了。
(2) 如果运行时间太长，那么想办法进行优化，使用分析方法就是优化方法之一。
(3) 如果可以使用分析方法替代一个计算方法，那么就以计算方法为比较的基础，用计算结果和分析结果互相进行验证。

对于遇到的绝大多数问题，我只用步骤 (1) 就能解决。

## 14.10 练习

本章练习的参考答案位于 chap14soln.py 中。

- 练习 14.1

5.4 节提到，成人体重大致符合对数正态分布。对此，一个可能的解释是，一个人每年增加的体重与当时的体重成比例。这样的话，成人体重就是大量因子的乘积。

$$w=w_0 f_1 f_2 \cdots f_n$$

其中 $w$ 是成人体重，$w_0$ 为出生时的体重，$f_i$ 为第 $i$ 年的体重增加因子。

乘积的对数是因子对数之和。

$$\log w=\log w_0+\log f_1+\log f_2+\cdots+\log f_n$$

根据中心极限定理，当 $n$ 很大时，$\log w$ 大致符合正态分布，因此 $w$ 符合对数正态分布。

为了对此现象进行建模，请为 $f$ 选择一个看似合理的分布，然后从出生时体重的分布中选择一个随机值，从 $f$ 的分布中选择一列因子，计算这些值的乘积，生成成人体重的一个样本。$n$ 需要为多大时，成人体重分布才会收敛到对数正态分布？

- 练习 14.2

14.6 节使用中心极限定理，得到在两个样本来自同一总体的原假设条件下，均值差异 $\delta$ 的抽样分布。

我们也可以使用这个分布，得到估计值与置信区间的标准误差，但结果只是大致正确。为了得到更精确的结果，我们应该计算当样本来自不同总体这一备责假设条件下，均值差异 $\delta$ 的抽样分布。

请计算这一分布，并用这个分布计算均值差异的标准误差和 90% 置信区间。

- 练习 14.3

在最近的一篇论文 [1] 中，Stein 等研究者调查了一项干预方法的效果，这一方法用于缓解学生工程小组中按性别分配任务的现象。

在干预前后，学生都参与了一项调查，对自己在课堂项目各方面所做的贡献进行评分，评分为 7 分制。

在干预前，男性学生对项目编程方面的打分高于女性学生。男性学生平均评分为 3.57，标准误差为 0.28；女性学生平均评分为 1.91，标准误差为 0.32。

请计算这一性别差异（均值差）的抽样分布，并检验这一差异是否统计显著。前面给出了估计均值的标准误差，因此无需知道样本规模就能算出抽样分布。

在干预实施后，这一性别差异变小了：男学生的平均评分为 3.44（SE 0.16）；女学生的平均评分为 3.18（SE 0.16）。请再次计算性别差异的抽样分布，并进行检验。

最后，请估计性别差异变化。这一变化的抽样分布如何？是否统计显著？

---

注 1："Evidence for the persistent effects of an intervention to mitigate gender-sterotypical task allocation within student engineering teams," Proceedings of the IEEE Frontiers in Education Conference, 2014.

# 作者介绍

**Allen Downey** 是富兰克林欧林工程学院计算机科学教授，曾执教于韦尔斯利学院、科尔比学院和加州大学伯克利分校。他在麻省理工学院获得计算机科学学士和硕士学位，在加州大学伯克利分校获得计算机科学博士学位。

# 封面介绍

本书封面上的动物是一只射水鱼。这种鱼能用嘴喷出水滴击落陆生昆虫和小动物，进行捕食。射水鱼有 7 种，在印度、菲律宾、澳大利亚和波利尼西亚都有分布。

射水鱼的口腔很深，从嘴到背鳍的空间形成一条直线，嘴部可以伸出，下颌外突。这种独特的嘴部构造就是射水鱼的秘密武器。射水鱼用舌头抵住口腔顶部的窄槽，收缩鳃盖，就可以激射出一股水流，射程可达 5 米。当射水鱼长到 2.5 厘米长时，就开始学习如何射水。最初小鱼们的命中率并不高，需要成群结队进行捕食，但随着经验的积累，技能会不断提高。

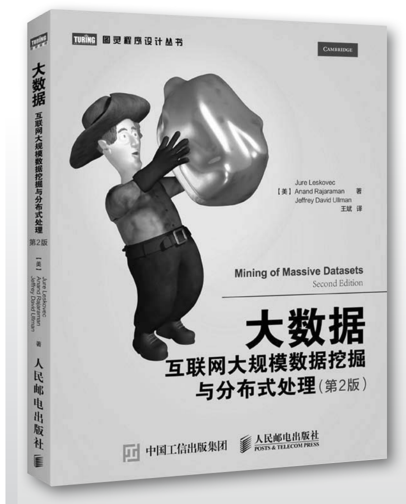
射水鱼的眼睛构造也得天独厚，视力奇佳，并能够在瞄准时修正水与空气之间的光线折射角度。射水鱼一旦发现猎物，就会转动眼睛，使目标的图像落入视野的特定位置。当昆虫落入攻击范围时，射水鱼经常会跃出水面，将其吞入口中。

射水鱼体型通常很小，只有 5~10 厘米长，但可以长到 40 厘米。很多水族馆都有射水鱼。

O'Reilly 封面中的许多动物都处于濒临灭绝的境地。它们对于我们这个世界都是至关重要的。如果你想更多地了解怎样帮助它们，请浏览 http://animals.oreilly.com 网站。

本书封面图片来自 *Dover*。

# 图灵最新重点图书




- **大数据权威著作全新升级版！**
- **第 1 版畅销 40000 册！**

本书源自作者在斯坦福大学教授的“海量数据挖掘”（CS246: Mining Massive Datasets）课程，第 1 版上市以来受到读者广泛欢迎和认可。本书以大数据环境下的数据挖掘和机器学习为重点，全面介绍了实践中行之有效的数据处理算法，是在校学生和相关从业人员的必备读物。

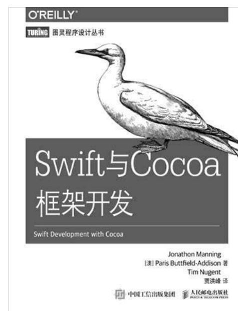
**大数据（第 2 版）**
书号：978-7-115-39525-2
定价：79.00 元



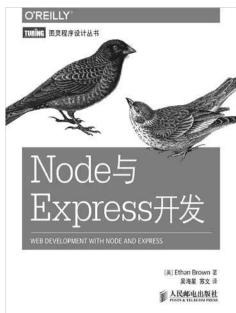

**数据科学实战**
书号：978-7-115-38349-5
定价：79.00 元




**命令行中的数据科学**
书号：978-7-115-39168-1
定价：49.00 元




**Swift 与 Cocoa 框架开发**
书号：978-7-115-39187-2
定价：89.00 元




**Node 与 Express 开发**
书号：978-7-115-38033-3
定价：69.00 元




**Java 8 函数式编程**
书号：978-7-115-38488-1
定价：39.00 元




**学习响应式设计**
书号：978-7-115-38973-2
定价：69.00 元

O'REILLY®

# 统计思维：程序员数学之概率统计(第2版)

现实工作中，人们常常需要用数据说话。可是，数据自己不会说话，需要人对它进行分析和挖掘才能找到有价值的信息。概率统计是数据分析的通用语言，是大数据时代预测未来的根基。如果你有编程背景，就能以概率和统计学为工具，将数据转化为有用的信息和知识，让数据说话。本书介绍了如何借助计算而非数学方法，使用Python语言对数据进行统计分析。

通过书中有趣的案例，你可以学到探索性数据分析的整个过程，从数据收集和生成统计量，到发现模式和检验假设。你还将探索概率分布、概率法则、可视化技术，以及其他许多工具和概念。

这一版内容较第1版有很多改动，并且新增了回归、时间序列分析、生存分析和分析方法章节，以丰富你的知识。

**通过学习本书，你将能够：**

- **编写测试代码深入理解概率论和统计学；**
- **运行实验检验统计行为特征，如生成服从各种分布的样本；**
- **通过模拟理解数学上艰涩的概念；**
- **学习贝叶斯估计等实用内容；**
- **用Python从大部分数据源导入数据，不依赖由统计工具清洗的格式化数据；**
- **用统计推理解读现实世界中的数据。**

“这是一本介绍Python数据分析的内容最为全面的书。数据分析师也可通过此书了解现代编程语言提供的工具，并提升技能。这还是一本优秀的现代统计学教程。”

——**Skipper Seabold**
*StatsModels*作者

**Allen B. Downey**是富兰克林欧林工程学院计算机科学教授，曾执教于韦尔斯利学院、科尔比学院和加州大学伯克利分校。在加州大学伯克利分校获得计算机科学博士学位。Downey已出版十余本技术书，包括*Think Python*、*Think Bayes*、*Think Complexity*等。

STATISTICS PROGRAMMING

封面设计：**Karen Montgomery** 张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/计算机数学

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版





ISBN 978-7-115-40108-3
定价：49.00元

# 看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks